



TRABALHO DE GRADUAÇÃO

**ACIONAMENTO REMOTO DE EQUIPAMENTOS
DOMÉSTICOS**

Autor:
Jefferson Rodrigues de Oliveira

Brasília 09 de Dezembro de 2016



**ENGENHARIA
MECATRÔNICA**
UNIVERSIDADE DE BRASÍLIA

TRABALHO DE GRADUAÇÃO

ACIONAMENTO REMOTO DE EQUIPAMENTOS DOMÉSTICOS

POR,

Jefferson Rodrigues de Oliveira

Relatório submetido como requisito parcial para obtenção
do grau de Engenheiro de Controle e Automação.

Banca Examinadora

Prof. Lelio Ribeiro Soares, UnB/ENE (Orientador)

Prof. Luis Filomeno Fernandes, UnB/FGA

Prof. Gerson H. Pfitscher, UnB/CIC

Brasília, 09 de Dezembro de 2016

FICHA CATALOGRÁFICA

JEFFERSON, OLIVEIRA

Acionamento remoto de equipamentos domésticos,

[Distrito Federal] 2016.

xvii, 74p., 297 mm (FT/UnB, Engenheiro, Controle e Automação, 2016). Trabalho de Graduação – Universidade de Brasília. Faculdade de Tecnologia.

1. Automação residencial

2. Acionamento remoto

3. *Shield* GSM SIM900

4. Arduino UNO

I. Mecatrônica/FT/UnB

II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

OLIVEIRA, Jefferson Rodrigues de, 2016. Acionamento remoto de equipamentos domésticos. Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT.TG-nº 041, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 74p.

CESSÃO DE DIREITOS

AUTOR: Jefferson Rodrigues de Oliveira.

TÍTULO DO TRABALHO DE GRADUAÇÃO: Acionamento remoto de equipamentos domésticos.

GRAU: Engenheiro

ANO: 2016

É concedida à Universidade de Brasília permissão para reproduzir cópias deste Trabalho de Graduação e para emprestar ou vender tais cópias somente para propósitos acadêmicos e científicos. O autor reserva outros direitos de publicação e nenhuma parte desse Trabalho de Graduação pode ser reproduzida sem autorização por escrito do autor.

Jefferson Rodrigues de Oliveira
QE 24, conjunto C, casa 27 – Guará.
71060-030 Brasília – DF – Brasil.

AGRADECIMENTOS

À minha família, em especial minha mãe por seu esforço pessoal para que eu pudesse me formar.

Ao meu orientador, pela paciência e colaboração para conclusão deste Trabalho de Graduação.

Jefferson Rodrigues de Oliveira.

RESUMO

Este trabalho constitui-se de um *app* escrito em linguagem Java para o Sistema Operacional Android, um Módulo GSM SIM900, alguns sensores (sensor de humidade DHT11, sensor de temperatura LM35, sensor de luminosidade LDR de 5 mm), a placa Arduino UNO (portando microcontrolador ATmega328) e um módulo de relé com 4 canais com optoacopladores. O *app* envia comandos e recebe (por meio de mensagens SMS) confirmação do *shield* GSM do acionamento dos equipamentos domésticos (representados por leds ou acionando os aparelhos em si – um umidificador e uma lâmpada), possibilitando o acionamento remoto de itens domésticos, bem como medindo variáveis importantes do ambiente a ser automatizado.

Palavras Chave: Automação residencial, acionamento remoto usando SMS, Shield GSM SIM900, Arduino Uno.

ABSTRACT

This Project is formed by an app wrote in Java language for operational system Android, a Shield GSM SIM900, some sensors (humidity sensor DHT11, temperature sensor LM35, luminosity sensor LDR), board Arduino Uno (bearing a microcontroller ATmega328) and a relay with 4 channels and optocouplers). The app sends commands and receives (via SMS messages) acknowledgment from the shield GSM SIM900 of actuation of domestic devices (represented by leds or triggering a device itself – a humidifier unit and a lamp), making possible the remote actuation of domestic devices, as measuring important variables of the place to be automated.

Keywords: Home automation; remote actuation via SMS; Shield GSM SIM900; Arduino Uno.

SUMÁRIO

LISTA DE SIGLAS E ABREVIATURAS	xi
CAPÍTULO 1 – INTRODUÇÃO	1
1.1 APRESENTAÇÃO DO PROBLEMA	1
1.2 OBJETIVOS DO PROJETO	1
1.3 METODOLOGIA	3
1.4 ESTRUTURA DA MONOGRAFIA	3
CAPÍTULO 2 – APRESENTAÇÃO DO PROBLEMA	5
2.1 BUSCA POR COMODIDADE E CONFORTO	5
2.2 TECNOLOGIAS USUAIS	5
2.2 VANTAGENS E DESVANTAGENS DO SIM900	6
CAPÍTULO 3 – REFERÊNCIAS TEÓRICAS	7
3.1 – MICROCONTROLADORES	7
3.1.1 – ARDUINO	7
3.2 – REDE GSM	8
3.3 – SMS	8
3.4 – SIM CARD	8
3.5 – LDR	9
3.6 – LM35	10
3.7 – DHT11	10
3.8 – Módulo GSM SIM900	11
3.9 – Módulo de Relés 5 V e 4 canais	12
CAPÍTULO 4 – DESCRIÇÃO DO MATERIAL E PROGRAMAS	14
4.1 – Arduino UNO	14
4.1.1 – Especificações	14
4.1.2 – Pinagem usada	15
4.2 – IDE Arduino	16
4.2.1 – Bibliotecas empregadas na IDE do Arduino	17
4.3 – Módulo GSM SIM900	18
4.4 – IDE Android Studio	19
4.4.1 – Bibliotecas da IDE Android Studio	20
4.5 – Componentes eletrônicos	Error! Bookmark not defined.
4.5.1 – LDR	20
4.5.2 – LM35	22
4.5.3 – DHT11	23
4.6 – Módulo de Relés	25
CAPÍTULO 5 – IMPLEMENTAÇÃO	26
Neste capítulo vamos abordar a fase prática deste trabalho em três partes:	
modelagem, programação e montagem do hardware.	26
5.1 – Modelagem	26
5.1 – Programação	28
5.1.2 – Programação IDE Arduino	30
5.1.3 – Programação IDE Android Studio	32
5.3 – Montagem do hardware	34
CAPÍTULO 6 – RESULTADOS OBTIDOS	35
6.1 – Resultados de software	36
6.2 – Resultados de hardware	40
6.3 – Problemas encontrados	43
CAPÍTULO 7 – CONSIDERAÇÕES FINAIS	44

7.1 – Conclusões	44
7.2 – Sugestões para trabalhos futuros	44
REFERÊNCIAS BIBLIOGRÁFICAS	45
APÊNDICE A – CÓDIGO FONTE IDE ARDUINO.....	47
APÊNDICE B – CÓDIGO FONTE IDE ANDROID STUDIO	52
ANEXO A – ESQUEMÁTICO ARDUINO	62

LISTA DE FIGURAS

Figura 2.1 – GSM remote control	5
Figura 2.2 – Módulo Ubee	6
Figura 3.1 – Arduino.....	7
Figura 3.2 – GSM SIM Card.....	9
Figura 3.3 – LDR.....	10
Figura 3.4 – LM35	10
Figura 3.5 – DHT11.....	11
Figura 3.6 – Módulo GSM SIM900	12
Figura 3.7 – Módulo de Relés 5 V e 4 canais.....	13
Figura 4.1 – Arduino Detalhada	14
Figura 4.2 – IDE Arduino	16
Figura 4.3 – Serial Monitor.....	18
Figura 4.4 – Esquemático Módulo GSM SIM900	18
Figura 4.5 – IDE Android Studio	19
Figura 4.7 – Montagem Virtual do LDR	21
Figura 4.8 – Esquemático LDR.....	22
Figura 4.9 – Montagem virtual do LM35 à Arduino	23
Figura 4.10 – Esquemático LM35.....	23
Figura 4.11 – Montagem virtual DHT11.....	24
Figura 4.12 – Esquemático DHT11.....	24
Figura 4.13 – Circuito do relé	25
Figura 5.1 – Modelagem do Trabalho completo	26
Figura 5.2 – Modelagem programa IDE Arduino	27
Figura 5.3 – Fluxograma IDE Android Studio	28
Figura 5.4 – Emulação IDE Android Studio	29
Figura 5.5 – Início do programa IDE Arduino	30
Figura 5.6 – Setup IDE Arduino	31
Figura 5.7 – Loop IDE Arduino 1ª parte.....	31
Figura 5.8 – Loop IDE Arduino 2ª parte	32
Figura 5.9 – Código-fonte IDE Android Studio 1ª parte	33
Figura 5.10 – Código-fonte IDE Android Studio 2ª parte	34
Figura 5.11 – Layout IDE Android Studio	34
Figura 5.12 – Montagem virtual do hardware.....	35
Figura 5.13 – Esquemático geral.....	35
Figura 6.1 – Setup do módulo GSM (Serial Monitor)	36
Figura 6.2 – Mensagem recebida ao iniciar o módulo.....	37
Figura 6.3 – Medidas dos sensores (Serial Monitor)	37
Figura 6.4 – Serial Monitor respondendo SMS.....	38
Figura 6.5 – Celular enviando e recebendo SMS ao módulo GSM	38

Figura 6.6 – Testes feitos no app 1ª parte.....	39
Figura 6.7 – Teste feitos no app 2ª parte.....	40
Figura 6.8 – Montagem do relé com o umidificador	40
Figura 6.9 – Módulo GSM e Arduino.....	41
Figura 6.10 – Módulo GSM, Arduino e sensores na protoboard	41
Figura 6.11 – Trabalho montado pronto para uso	42
Figura 6.12 – Esquema montado e acionado	42
Figura 6.13 – Esquema montado e desligado.....	43

LISTA DE TABELAS

Tabela 4.1 – Características resumidas da Arduino Uno	15
Tabela 4.2 – Numeração da pinagem usada no Trabalho.....	15
Tabela 4.3 – Legenda da Figura 4.2.....	16
Tabela 4.4 – Legenda da Figura 4.5	20

LISTA DE SIGLAS, ABREVIATURAS E ACRÔNIMOS

<i>AT</i>	Hayes AT Commands (Comandos AT Hayes)
<i>cm</i>	Centímetros
<i>GSM</i>	Global System for Mobile Communications (Sistema Global para Comunicações Móveis)
<i>IDE</i>	Integrated Development Enviroment (Ambiente Integrado de Desenvolvimento)
<i>MHz</i>	Mega Hertz
<i>USB</i>	Universal Serial Bus (Porta Serial Universal)
<i>V</i>	Volts
<i>mV</i>	mili Volt
<i>A</i>	Ampere
<i>RAM</i>	Random Access Memory (Memória de acesso aleatório)
<i>PWM</i>	Pulse Width Modulation (Modulação por largura de pulso)
<i>EPROM</i>	Erasable Programmable Read-Only Memory (Memória apagável programável somente de leitura)
<i>LED</i>	Light Emissor Diode (Diodo emissor de luz)
<i>mA</i>	mili Ampere

CAPÍTULO 1 – INTRODUÇÃO

1.1 APRESENTAÇÃO DO PROBLEMA

A vida moderna, com suas mudanças em ritmo acelerado, deixa cada vez menor o tempo disponível das pessoas. Considerando este fato, o engenheiro de controle e automação pode contribuir para diminuir a atenção e o tempo que as pessoas gastam para gerir seus bens no contexto doméstico.

“... a automação robótica permite maior qualidade de vida, reduz o trabalho doméstico, aumenta o bem-estar e a segurança, racionaliza o consumo de energia e, além disso, sua evolução permite oferecer continuamente novas aplicações.” [1]

Ademais, o conforto é outro fator que também aumenta quando se dispõe de automação. Muitas vezes a automação é procurada com este fim somente. Afinal, qualidade de vida está entre os itens mais valorizados da atualidade na sociedade brasileira.

Enfim, a automação doméstica ou domótica (termo usado no livro *Automação Doméstica*, de José Roberto Muratori e Paulo Henrique Dal Bó) ensejou uma nova oportunidade de mercado, criando um campo de mercado e novas profissões, das quais cita-se o engenheiro de automação, o integrador de residências (profissional responsável por conciliar vários dispositivos domésticos permitindo o controle e monitoramento do lar de maneira integrada, um generalista em automação) e outras profissões relacionadas a automação doméstica.

“Hoje no Brasil temos uma alta demanda pelo “integrador de residências”, pois essas tecnologias ainda não possuem o conceito plug-and-play. Necessitamos de pessoas qualificadas e treinadas para projetar, construir, instalar e programar tais equipamentos. Em outros países, essa situação é diferente, pois existem profissionais com equipes especializadas em cada parte do projeto, fazendo com que os resultados apareçam de forma mais rápida.” [2]

1.2 OBJETIVOS DO PROJETO

O trabalho pretende criar um esquema básico e comerciável de automação doméstica. Em especial, o acionamento automático e remoto de aparelhos domésticos, bem como integrar a comunicação com a telefonia móvel via aplicativo Android e medir variáveis importantes no monitoramento de uma residência.

Para isto, o aplicativo criado (a ser detalhado neste Trabalho) envia mensagens segundo um protocolo de comunicação com o Módulo *GSM SIM900*. O módulo *GSM SIM900*,

por sua vez, se comunica com o *Arduino UNO*. O *Arduino*, finalmente, processa as informações e efetua o acionamento dos dispositivos.

Esse modelo pretende facilitar o acionamento de dispositivos domésticos remotamente devido à interface com o aplicativo *Android*. Além disso, com a interface amigável, monitorar dispositivos importantes pelo celular.

Importante também ressaltar a simplicidade do programa *Android* que foi desenvolvido, de modo a deixar aberto para pequenas mudanças em seu código. O código foi feito na linguagem Java e por meio da *IDE Android Studio*, gratuitamente distribuída pela *Google*.

1.3 METODOLOGIA

Com a intenção de criar um esquema que possua comunicação robusta, a ideia foi implementar o módulo que dependesse de tecnologia *SMS*. Para que este trabalho seja também mais amigável com o usuário, um programa para celular foi também pensado. O *Shield GSM* usa o protocolo *GSM* e foi programado por meio de comandos *AT* desenvolvidos na *IDE* da *Arduino* (software distribuído gratuitamente pela *Arduino*).

Durante a montagem da *Shield GSM* foram feitas algumas medições para avaliar o funcionamento do módulo integrado com o *Arduino*, pois a princípio o módulo teve problemas para funcionar com chips *GSM* das operadoras Oi e Tim. Entretanto, funcionou perfeitamente com o chip *GSM* Claro, devido a problemas de compatibilidade do fabricante do módulo com chips das outras operadoras.

Os sensores foram todos testados isoladamente, para entendimento de suas bibliotecas e para definir os circuitos (resistores e componentes eletrônicos) que seriam adequados para sua instalação. Analogamente, o relé foi testado para verificar seu funcionamento com apenas um canal, o qual respondeu adequadamente. Em seguida, incorporados para comunicação com o *Shield* e com o aplicativo.

A elaboração do aplicativo foi baseada em programas simples que enviam *SMS* disponíveis em tutoriais do site *stackoverflow.com*. Em seguida, os demais incrementos do aplicativo foram escritos usando os conhecimentos básicos de linguagem em C e mais tutoriais disponíveis no referido site *stackoverflow.com*.

Todos os circuitos foram montados sobre uma placa protoboard de 640 pinos sem base. Os resistores usados foram os de 220 Ohms e 10 kOhms. As conexões feitas por fios simples coloridos de pontas pinadas.

1.4 ESTRUTURA DA MONOGRAFIA

Esta monografia é dividida em 6(seis) capítulos. O primeiro, introdutório, aborda superficialmente a relação entre automação residencial e o engenheiro. Além disso, também aborda a metodologia empregada.

O segundo capítulo aborda o problema, bem como suas motivações e oportunidades que inspiraram o desenvolvimento desse Trabalho de Graduação. Explica também a escolha das ferramentas usadas e sua relação com o mercado.

O referencial teórico, terceiro capítulo, traz embasamento teórico necessário para entender esta monografia, considerando que o leitor tenha algum conhecimento de tecnologias.

O quarto capítulo versa sobre os componentes usados no projeto. O detalhamento vai desde o *hardware* até o *software*, abordando as tecnologias usadas.

O quinto capítulo, implementação, traz os detalhes da implementação, o passo-a-passo do trabalho e as peculiaridades apresentadas durante o desenvolvimento do projeto.

Os resultados são apresentados no sexto capítulo, mostrando-se as simulações e testes feitos para observar todos os dispositivos principais usados no Trabalho num ambiente real. Além das dificuldades enfrentadas ao desenvolver este projeto.

Ao final, as considerações finais apresentam conclusões do trabalho bem como sugestões para futuros trabalhos a surgir a partir deste.

CAPÍTULO 2 – APRESENTAÇÃO DO PROBLEMA

Este capítulo detalha a apresentação já abordada na introdução desta monografia e motivações para a escolha do tema.

2.1 BUSCA POR COMODIDADE E CONFORTO

Observando as tendências de mercado, propusemos a elaboração deste Trabalho. A busca por conforto, economia de energia, redução do trabalho doméstico, segurança de poder ter acesso permanente à sua residência e também poder monitorar serviu como motivação.

Cada vez mais pessoas estão dispostas a automatizar suas casas como forma de aumentar a segurança e o conforto de seus lares. Além disso, com a popularização destes métodos, esta tendência deve baratear o preço final do produto e favorecer o desenvolvimento deste seguimento.

2.2 TECNOLOGIAS USUAIS

“O dispositivo é o *GSM Remote Control – 2 IN and 2 OUT*. Possui PIC18F46K20-I/PT como microcontrolador. É acionado remotamente via SMS e possui funcionalidade próxima ao *Shield GSM SIM900*.”[3]



Figura 2.1 – GSM remote control

Também no segmento de automação residencial, porém de acionamento remoto a uma menor distância, é bastante comum o uso de aparelhos que usam protocolo *Zigbee*. Esse protocolo conta com diversas marcas, dentre elas o módulo *Ubee*. Esse método é o mais comum em integração de dispositivos residenciais no contexto da automação residencial.



Figura 2.2 – Módulo Ubee

2.2 VANTAGENS E DESVANTAGENS DO SIM900

O *Shield GSM SIM900* possui várias vantagens práticas, como por exemplo, ser de fácil acesso para quem procura uma automação rápida. Além disso, ele é capaz de receber ligações (capacidade que pode ser explorada em outras formas de acionamento remoto).

Possui vasta documentação na Internet, o que facilita bastante na etapa de implementação de projeto.

Todavia, o *Shield GSM SIM900* não faz leitura do estado em que se encontram os dispositivos, a menos que sejam criados *flags* enviados de tempos em tempos ou se implemente outros dispositivos de medida de estado em conexão com o módulo SIM900.

CAPÍTULO 3 – REFERÊNCIAS TEÓRICAS

Este capítulo se dispõe a esclarecer alguns termos usados nesse projeto. Embora esses conceitos sejam amplos, ambíguos e mais profundos, esse referencial vai explorá-los.

3.1 – MICROCONTROLADORES

“Os microcontroladores são circuitos integrados de baixo custo que contém em sua síntese: memória programável somente para leitura, que armazena permanentemente as instruções programadas; memória *RAM*, que trabalha armazenando “variáveis” utilizadas pelo programa; *CPU*, que interpreta e executa comandos desse programa. Existem também dispositivos de entradas e saídas, que tem a finalidade de controle de dispositivos externos ou de receber sinais pulsados de chaves e sensores.” [3]

O microcontrolador é uma pequena plataforma de processamento usada em tarefas que exigem pouco processamento e pouca memória. Dentre essas tarefas destacam automações industriais e residenciais, controle de dispositivos e outras tarefas que consomem pouco processamento comparativamente a um programa elaborado como um Sistema Operacional.

3.1.1 – ARDUINO



Figura 3.1 – Arduino

Grande parte da motivação pela escolha do Arduino foi sua praticidade, grande documentação disponível na internet, bem como sua ampla utilização em projetos similares de automação residencial. Trata-se de um ambiente multiplataforma que suporta Linux, Mac OS e Windows. Possui uma *IDE* (Integrated Development Environment, Ambiente Integrado

de Desenvolvimento em português) de programação do tipo *Processing*, um ambiente de desenvolvimento amigável e gratuito, fornecido pelo fabricante e aprimorado por usuários.

Sua comunicação com o computador é feita por cabo *USB* compatível com a maioria dos computadores modernos.

3.2 – REDE GSM

A sigla *GSM* significa *Global System for Mobile Communications* ou Sistema Global para Comunicações Móveis e é o padrão mais utilizado para comunicação entre dispositivos móveis de intercomunicação celular. Ultrapassam a quantia de 6 bilhões de pessoas ao redor do mundo.

Sua área de cobertura satélite 90% da população mundial e seu modo *roaming* (ou intinerância, é um termo usado para o serviço de interconectividade de redes para os usuários que saem de suas áreas de cobertura geográfica) permite que o usuário estenda sua área original. É uma tecnologia criada para permitir a comunicação via dados e voz.

3.3 – SMS

SMS é sigla para *Short Message Service*, em tradução livre significa Serviço de Mensagem Curta. É um serviço muito utilizado para envio de textos curtos entre aparelhos celulares. É um serviço rápido e eficiente, compatível com a tecnologia *GSM*.

É composto por 160 letras. O usuário pode usar 140 caracteres diferentes para compor a mensagem. Após composta a mensagem, o usuário a envia e ela segue pelo sistema de satélite da operadora.

Embora seja uma tecnologia relativamente obsoleta de comunicação entre aparelhos celulares, ela conta com uma cobertura maior e usa uma tecnologia mais simples. Por essa razão, sua viabilidade é tão presente quanto é o sinal da operadora, ao contrário de outros métodos que dependem de sinal de internet para sua comunicação.

3.4 – SIM CARD

O *SIM card* é um pequeno circuito impresso usado para identificar cada usuário da tecnologia *GSM*. Além dessa função principal, armazena dados importantes e serve também para controlar variáveis do usuário.

SIM é sigla para *Subscriber Identity Module*, em português Módulo de Identificação do Assinante. Fisicamente, é uma pequena placa de plástico com dimensões 25 x 15 mm, com um pequeno chanfro em uma das extremidades.



Figura 3.2 – GSM SIM Card

3.5 – LDR

O LDR (*Light Dependent Resistor*, ou Resistor Dependente de Luz), é um componente eletrônico semelhante a um resistor de resistência variável, exceto que sua resistência muda de acordo com a luminosidade que incide sobre sua superfície. Mais especificamente, o que se altera nesse dispositivo é sua resistividade, isto é, sua característica física de resistir à tensão. [3]

É composto por um semicondutor (sulfeto de cádmio (CdS) ou sulfeto de chumbo (PbS)) com uma camada protetora para proteger do oxigênio (O₂) presente no ar. Geralmente é usado em sensores de presença residenciais, contagem industrial ou alarmes.

Suas dimensões são de 5 mm ou de 7 mm (a parte superior) e seu corpo em torno de 2 cm.



Figura 3.3 – LDR

3.6 – LM35

O LM35 é um sensor de temperatura composto por um circuito integrado contendo um transistor de encapsulamento TO-92 de 3 pinos, bastante sensível, preciso e barato, por esse motivo bastante comercializado.

Esse sensor possui sensibilidade de $10\text{mV}/^{\circ}\text{C}$, ou seja, para cada grau Celsius variado, ele fornece uma saída variando em 10mV. Sua faixa de operação vai dos -55°C aos 150°C , com precisão de $0,5^{\circ}\text{C}$.



Figura 3.4 – LM35

3.7 – DHT11

O DHT11 é um sensor de umidade relativa e temperatura com saída digital. Possui um sensor de umidade do tipo HR202 e como sensor de temperatura um termistor NTC

(semicondutor sensível à temperatura, com coeficiente de resistividade negativo com o aumento da temperatura).

Possui internamente um microcontrolador de 8 bits para tratar o sinal que emite. São calibrados de fábrica e gravados no programa que executa o microcontrolador. Possui tamanho pequeno, consome pouca corrente e encapsulamento simples (o que aumenta sua robustez).

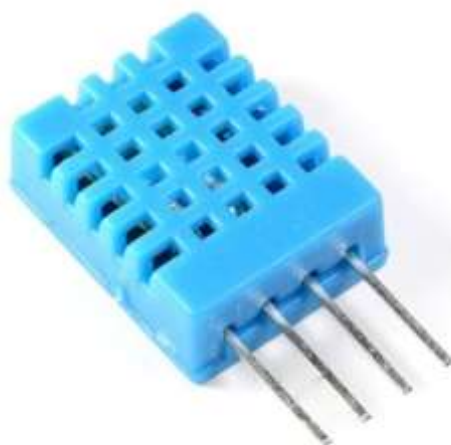


Figura 3.5 – *DHT11*

3.8 – Módulo *GSM SIM900*

O Módulo *GSM SIM900* é uma placa acoplável ao *Arduino UNO* e à *Arduino MEGA* capaz de enviar e receber mensagens eletrônicas e fazer ligações de voz (possui entrada de áudio para permitir a entrada e saída de áudio). Funciona em 4 bandas de frequência comuns (850, 900, 1800 e 1900 MHz).

É controlada por comandos *AT* (*GSM 07.07*, *07.05* e *SIMCOM* com comandos *AT* aprimorados). Na conexão com o *Arduino UNO*, a comunicação é feita por pinos que enviam esses comandos. No caso do trabalho em questão, é feita através dos pinos 2 e 3.



Figura 3.6 – Módulo GSM SIM900

3.9 – Módulo de Relés 5 V e 4 canais

Um Módulo de Relés 5 V e 4 canais é um agrupamento de 4 relés em uma única placa para serem controlados por uma única fonte e interagir com 4 acionamentos. Seu princípio físico de funcionamento é o eletromagnetismo que aciona uma chave e fecha um contato, através de alimentações de 5 V.

“Este dispositivo promove a interação entre aparelhos eletrônicos (5 V) e aparelhos elétricos (energizados na faixa de 30 V e 10A até 250 V e 10A). Deste modo, é possível acionar aparelhos de maior porte a partir de comandos digitais.” [10]



Figura 3.7 – Módulo de Relés 5 V e 4 canais

CAPÍTULO 4 – DESCRIÇÃO DO MATERIAL E PROGRAMAS

4.1 – Arduino UNO

A *Arduino UNO* é uma das últimas versões da placa *Arduino*, feita para pequenos projetos. Tem se mostrado prática e útil, com poucas limitações. Robusta, é adequada para grande diversidade de projetos do dia-a-dia.

O esquemático da *Arduino UNO* será apresentado na parte de anexos deste trabalho.

4.1.1 – Especificações

A *Arduino UNO*, feita pra trabalhar com tensões compatíveis com as usadas na eletrônica (5V como voltagem de operação) para comunicação. Na descrição do produto, recomenda-se uma tensão de alimentação na faixa de 7V – 12V, para que possa alimentar seus circuitos, porém tolera uma alimentação na faixa entre 6V como mínima e 20V como tensão máxima de alimentação. [6]

Possui um total de 28 pinos. Dezesesseis desses pinos são para entrada ou saída digitais. Mais 6 pinos para entradas analógicas. Os 6 pinos restantes são para alimentação de circuitos acessórios.

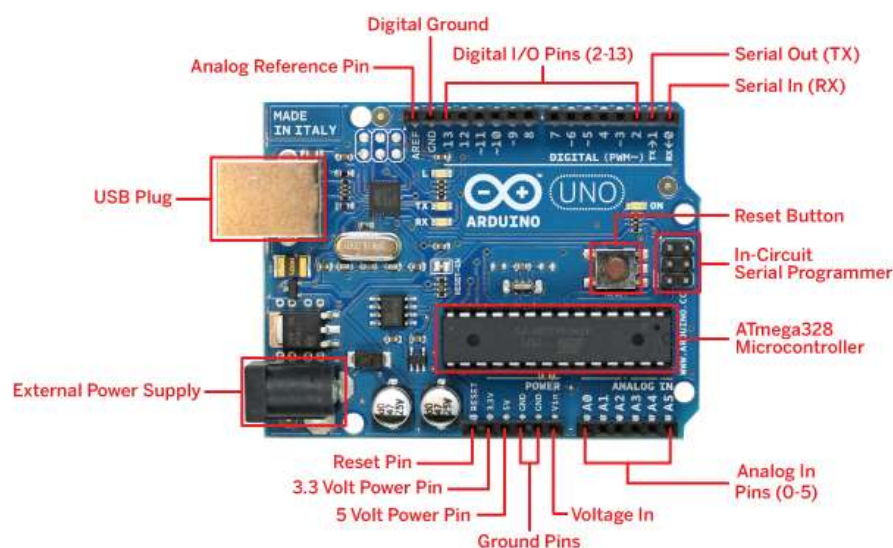


Figura 4.1 – Arduino Detalhada

A *Arduino UNO* possui uma entrada *USB* para comunicação em substituição a entrada serial presentes em modelos anteriores da *Arduino*. Seu microcontrolador de 8 bits é um ATMEL ATMEGA328, capaz de processamentos para pequenas quantidades de dados.

Seus pinos 3, 5, 6, 9, 10 e 11 podem ser usados como saídas *PWM* de 8 bits de resolução, ou seja, podem enviar sinais binários que variam de 0 V a 5 V em submúltiplos de 256, usando a função *analogWrite()* pela interface de programação. Além de entradas analógicas que serão usadas para ler dados dos sensores empregados no trabalho.

Seguem as principais características da placa para usuários iniciantes, disponíveis no próprio site da Arduino.

Tabela 4.1 – Características resumidas da *Arduino Uno*

Item	Descrição
Voltagem de alimentação recomendada	7 V a 12 V
Voltagem de alimentada tolerada	6 V a 20 V
Voltagem de operação	5 V
Microcontrolador	ATMEL ATMEGA328
Pinos de entrada e saída digitais	14 (0 ao 13)
Pinos de entrada analógica	6 (A0 ao A5)
Corrente contínua de saída e entrada	20 mA
Frequência de operação	16 MHz
Memória Programável (<i>Flash</i>)	32 KB

O pino 13 é usado para testes da placa em usos iniciais (relatado em “LED_BUILTIN”). A *Arduino UNO* possui também pinos para alimentação e tensão de referência (terra), detalhados na Figura 4.1, usados para alimentação de circuitos que são integrados na implementação de esquemáticos variados.

4.1.2 – Pinagem usada

A tabela a seguir detalha os pinos usados, bem como sua função no trabalho.

Tabela 4.2 – Numeração da pinagem usada no Trabalho

Pino	Uso
<i>GND</i>	Referência para todos os sensores
5 V	Alimentação dos sensores e do <i>led</i>
2 e 3 (Digitais)	Comunicação serial com o <i>shield</i> SIM900
7(Digital)	Alimentação para o <i>led</i>
8(Digital)	Acionamento do relé(umidificador)
13(Digital)	Comunicação com o Chip
11(Digital)	Leitura de umidade (<i>DHT11</i>)
0(Digital)	Leitura de luminosidade(<i>LDR</i>)

4.2 – IDE Arduino

A *IDE Arduino* é uma interface de desenvolvimento escrita em linguagem JAVA, própria para programação e *upload* de programas para a placa *Arduino*. Possui uma parte para preparação de bibliotecas e conexões com outros módulos ou configurações – *void setup()* – além da parte de execução do algoritmo – *void loop()*.

A interface *IDE Arduino* é responsável por traduzir o programa escrito em C para a linguagem de máquina executada no microcontrolador da *Arduino* — ATMEGA328. A seguir, apresenta-se a descrição das principais funcionalidades do programa.

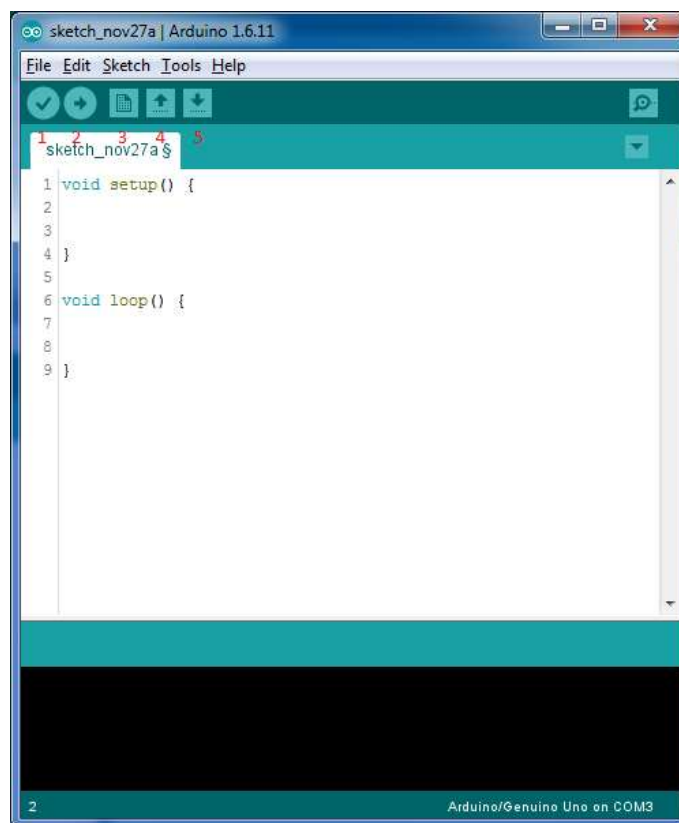


Figura 4.2 – IDE Arduino

A tabela a seguir explana os números em vermelho da figura acima.

Tabela 4.3 – Legenda da Figura 4.2

Número	Função
1	<i>Verify</i> – Verifica e compila o programa
2	<i>Upload</i> – Sobe o programa para a Arduino
3	<i>New</i> – Cria um arquivo novo

4	<i>Open</i> – Abre programas criados
5	<i>Save</i> – Salva o programa em confecção

4.2.1 – Bibliotecas empregadas na *IDE Arduino*

Ao usar a *IDE da Arduino* é possível encontrar bibliotecas que facilitam a programação ao permitir abstrair toda uma cadeia de comandos para se manipular e interpretar componentes ligados à placa.

Como primeira e mais importante biblioteca deste projeto, temos a “sms.h”, encontrada na página: github.com/MarcoMartines/GSM-GPRS-GPS-Shield. Esta biblioteca, introduzida pelo comando “#include sms.h” ao começo do programa, faz com que substituamos os comandos AT, tais como “AT+CMGF=1” (que impomos o modo de envio de mensagem de texto para o módulo), “AT+CPAS” (retorna o status de atividade do módulo) por macros como “gsm.begin();” que inicializa o módulo e o deixa pronto para uso.

Além desse comando, temos outros dois usados para enviar e receber *SMS*: o “sms.SendSMS(char *numero, char *mensagem);” que recebe o número do telefone e a mensagem a ser enviada; “sms.GetSMS(char posição, char *numero, int digitosdonumero, char *mensagem, int tamanhodomensagem)” que recebe a posição em que está a mensagem a ser lida, o número do telefone que enviou, o tamanho do número, a mensagem e o número de caracteres da mensagem.

Além dessa biblioteca, temos a biblioteca do DHT11 – DHT.h (“#include DHT.h” - github.com/adafruit/DHT-sensor-library) que permite abstrairmos a matemática por trás do processamento após as leituras do sensor DHT11, em especial a conversão de dados analógicos em digitais. Em especial, o comando “dht.readHumidity();” para leitura da umidade. Embora o sensor seja capaz de medir temperatura também, a leitura de temperatura foi feita pelo LM35 devido a oscilações na leitura medida pelo DHT11 e a boa precisão do LM35.

Outra biblioteca importante, em especial na fase de testes, foi a “SoftwareSerial.h” (incluída pela diretiva “#include <SoftwareSerial.h>”, encontrada nativamente na *IDE da Arduino*). Essa biblioteca permite visualizar o desenvolvimento do programa por meio do *Serial Monitor* a ser ilustrado a seguir, na medida em que exhibe a preparação do programa e mostra também a execução dos comandos em tempo real na placa Arduino.

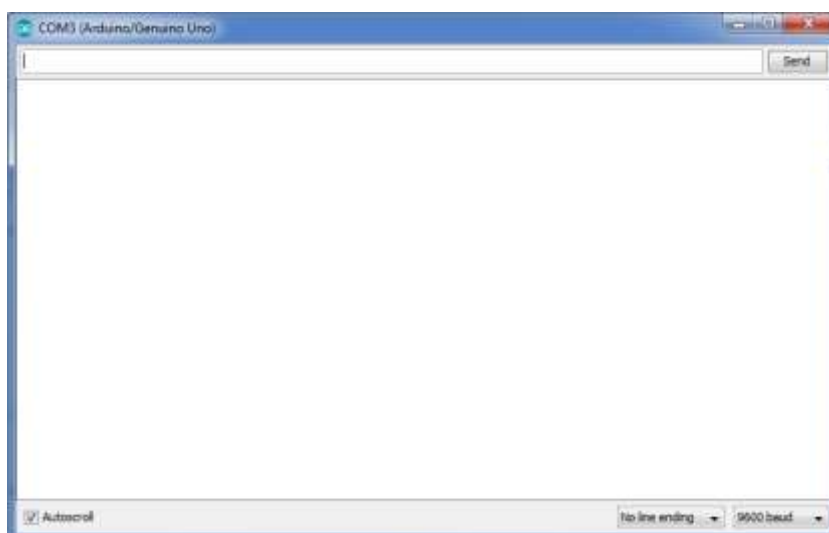


Figura 4.3 – Serial Monitor

4.3 – Módulo GSM SIM900

O módulo *GSM SIM900*, ou shield *GSM*, é uma placa que se acopla à Arduino de modo a estender sua funcionalidade, qual seja, comunicação *GSM*. O módulo deve ser alimentado por uma tensão de 5 V e possui um processador próprio para executar as tarefas solicitadas via comandos AT pela Arduino. [5]

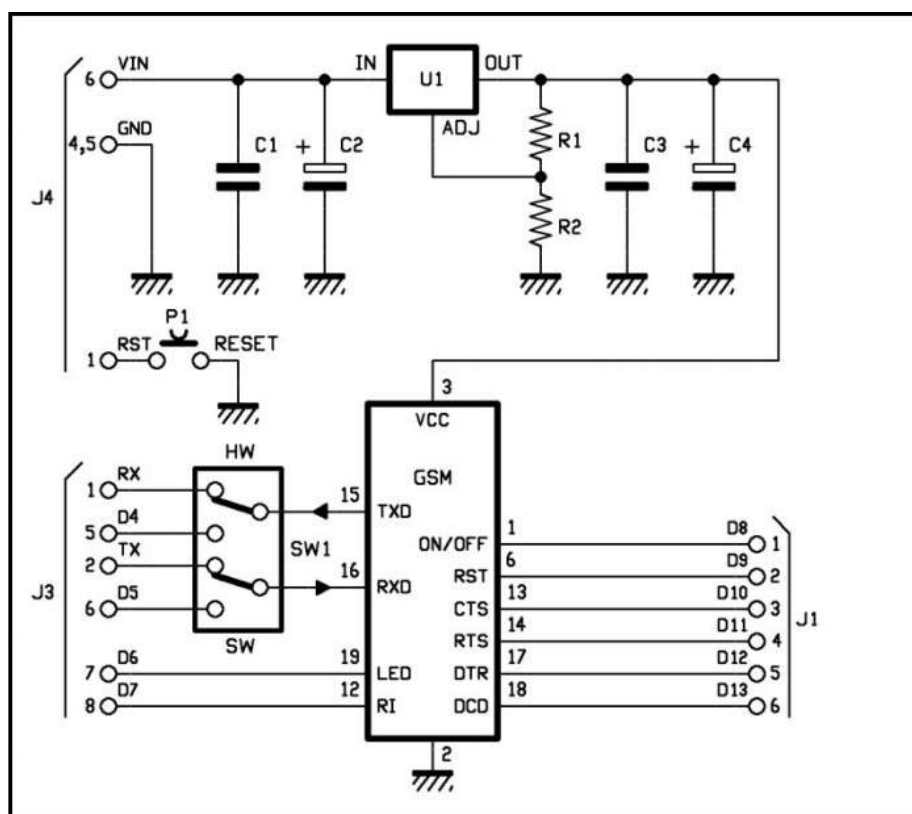


Figura 4.4 – Esquemático Módulo GSM SIM900

4.4 – IDE Android Studio

A *IDE Android Studio* é a interface de desenvolvimento para programas para o Sistema Operacional Android fornecida gratuitamente pela *Google*. É um programa completo e bastante facilitado para qualquer um com noções de programação, ou mesmo um total novato consiga fazer programas simples para Android.

O Android Studio compõe-se de três partes básicas responsáveis pela elaboração do programa em si, as bibliotecas a serem usadas e módulos de simulação dos programas feitos.

Quanto à programação, existem três partes que compõe o programa: o *manifest* que contém as permissões para acessar partes do Sistema Operacional do celular, bem como outros componentes do celular como o GPS, por exemplo; o Java que contém o código-fonte que comporá o aplicativo; o *res* que contém a parte interativa do programa como o layout, imagens usadas, botões etc. [9]

Após configuradas as especificações sobre como será seu programa, o Android Studio exibe a seguinte tela para a confecção e simulação de programas. Abaixo a legenda da figura representada em uma tabela.

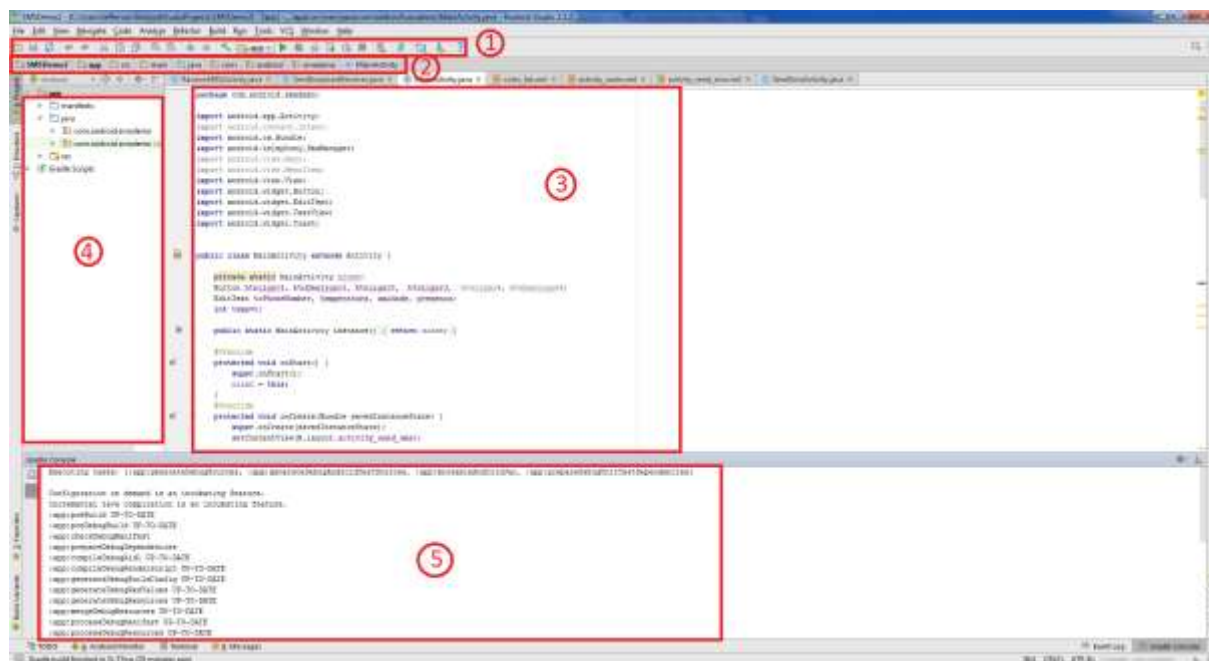


Figura 4.5 – IDE Android Studio

Tabela 4.4 – Legenda da Figura 4.5

Número	Significado
1	Barra de Ferramentas
2	Navegação entre as partes do programa
3	Janela do editor de código e <i>layout</i>
4	Janela de navegação em hierarquia
5	Compilador e reporte de erros

4.4.1 – Bibliotecas da IDE Android Studio

Com a função de facilitar a programação e abstrair diversos comandos minuciosos nessa interface tão complexa, foram usadas bibliotecas prontas, que podem ser baixadas diretamente pelo programa Android Studio. [11]

Foram usadas 8 bibliotecas diferentes na confecção do aplicativo. Explanaremos a função de cada uma das funções:

- “android.telephony.Smsmessage” responsável pelo envio de mensagens SMS pelo aplicativo, permitindo a comunicação com módulo GSM;
- “android.content.BroadcastReceiver” responsável pelo recebimento de SMS pelo programa;
- “android.widget.Button” responsável pelos botões usados nas funções
- “android.widget.Toast” responsável pela mensagem rápida ao ser enviada e recebida uma mensagem;
- “android.widget.EditText” responsável pelos campos que possuem textos editáveis;
- “android.widget.TextView” responsável pelos campos que contém textos fixos, que não podem ser editados;
- “android.view.View” biblioteca que contém funções de visualização no *Android Studio*, essencial para qualquer programa com interface gráfica;
- “android.os.Bundle” biblioteca que inicializa as *activity* no *Android Studio*;

4.5 – Componentes eletrônicos

Nesta seção trataremos dos componentes eletrônicos usados neste trabalho. Na ordem serão o LDR (seção 4.5.1), LM35 (seção 4.5.2), DHT11 (seção 4.5.3).

4.5.1 – LDR

O Resistor Dependente de Luz ou LDR é um resistor cuja resistividade varia conforme se incide radiação eletromagnética visível em sua superfície. O LDR de sulfeto de cádmio (CdS) opera com resistência entre $400\ \Omega$ a $10\ \text{k}\Omega$ quando em total escuro e exposto ao limiar de luminosidade (potência de $90\ \text{mW}$), respectivamente. Apesar de não possuir resistência nula quando no escuro, esse sensor foi associado em paralelo a uma resistência de $10\ \text{k}\Omega$ para ter uma referência quando haver corrente percorrendo por ele. Este circuito foi alimentado com uma tensão de $5\ \text{V}$ e tem como saída tensões que variam analogicamente entre $0.4\ \text{V}$ a $5\ \text{V}$.

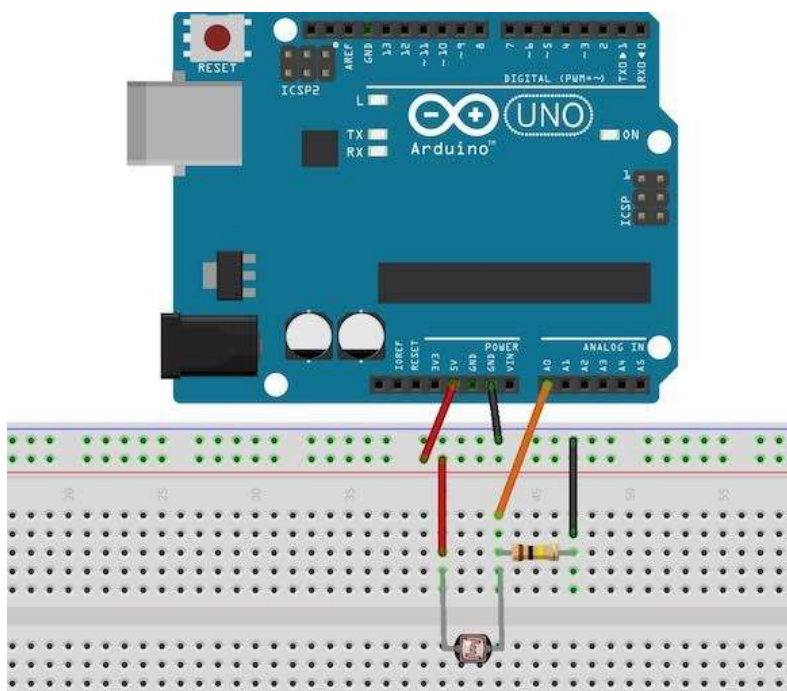


Figura 4.7 – Montagem Virtual do LDR

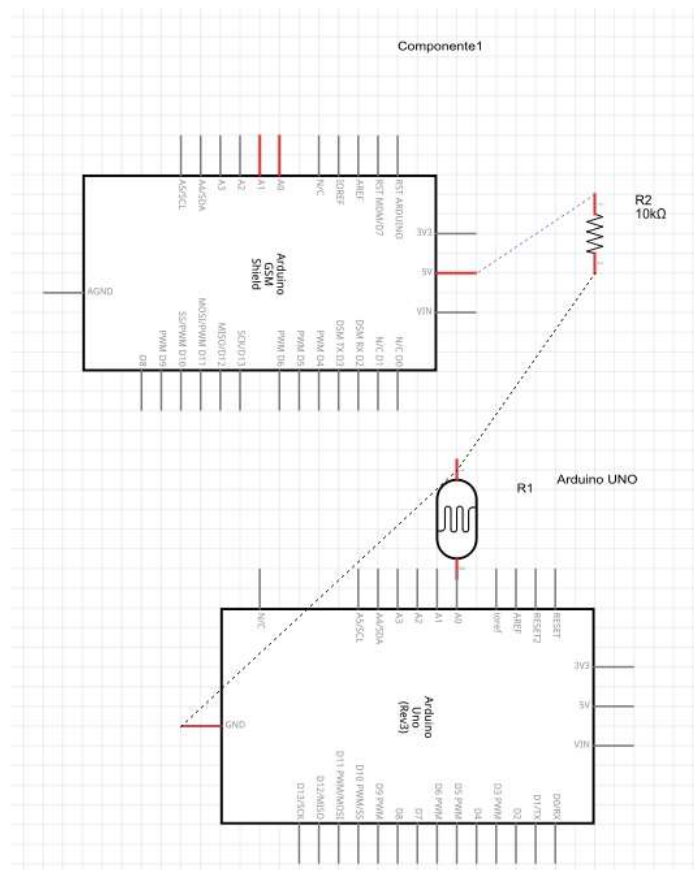


Figura 4.8 – Esquemático LDR

4.5.2– LM35

O LM35 é um sensor de temperatura usado comumente associado a circuitos eletrônicos pois sua entrada é 5 V fornecida, como fornecida pela Arduino. Como saída, emite um sinal analógico de 0 V correspondente a 0° C com variações e 1,5 V correspondente a 150° C, embora, com resolução de 10 mV/°C. Portanto, basta fazer transformação de binário (0 a 1023) para ° C no algoritmo da IDE Arduino. Esta transformação será tratada mais adiante. [8]

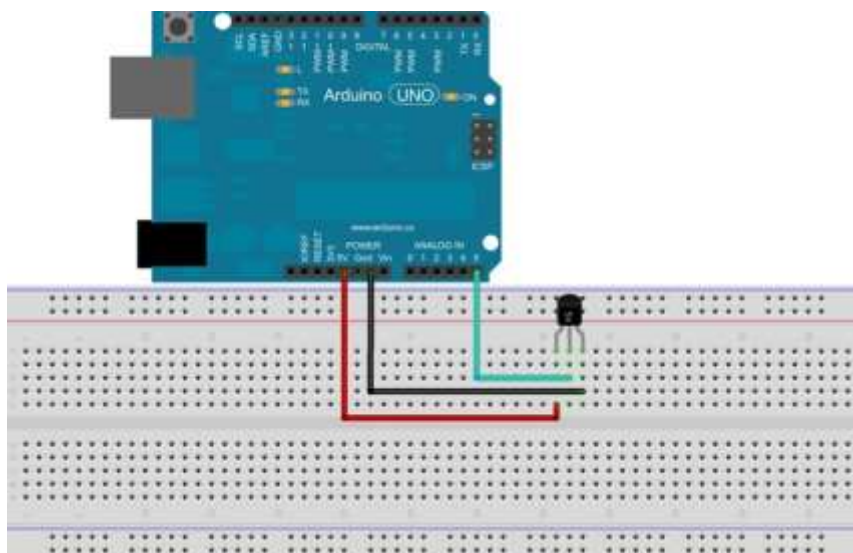


Figura 4.9 – Montagem virtual do LM35 à Arduino

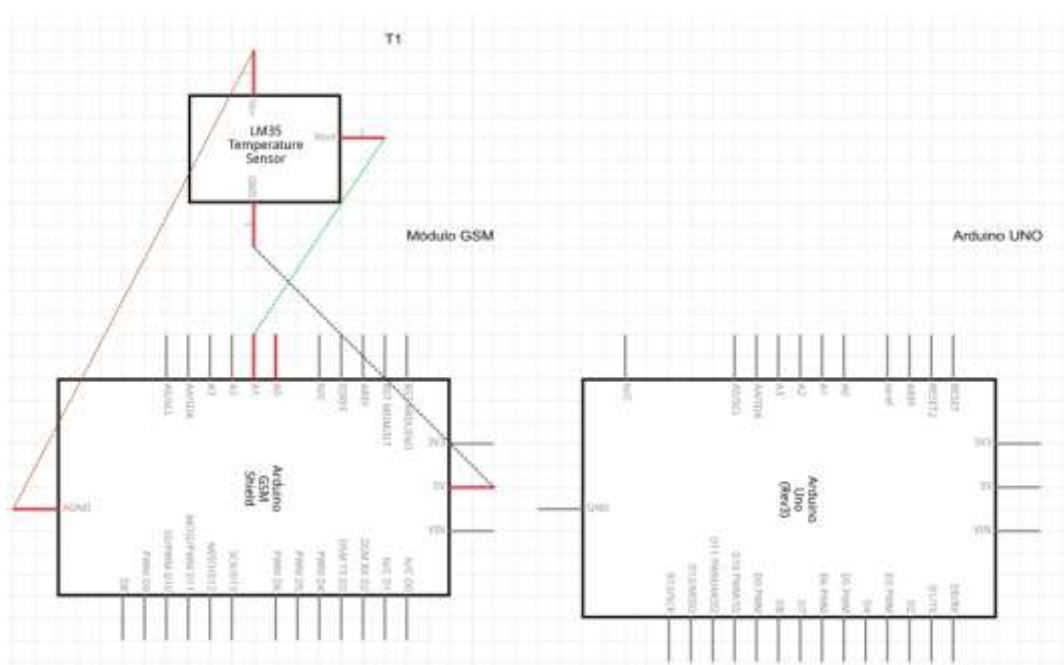


Figura 4.10 – Esquemático LM35

4.5.3– DHT11

O sensor de umidade e temperatura DHT11 é capaz de realizar medidas de temperatura de 0° C a 50° C com resolução de 1° C e acurácia de $\pm 1^\circ$ C. Sua medida de umidade tem alcance entre 20 %RH e 80 %RH medidas a 25° C. Sua saída analógica é processada primeiramente por seu próprio microcontrolador e repassada digitalmente para a Arduino. No entanto, faz-se necessário ligar sua saída em série com um resistor para limitar a corrente. As duas figuras a seguir apresentam a montagem virtual e o esquemático do sensor. [7]

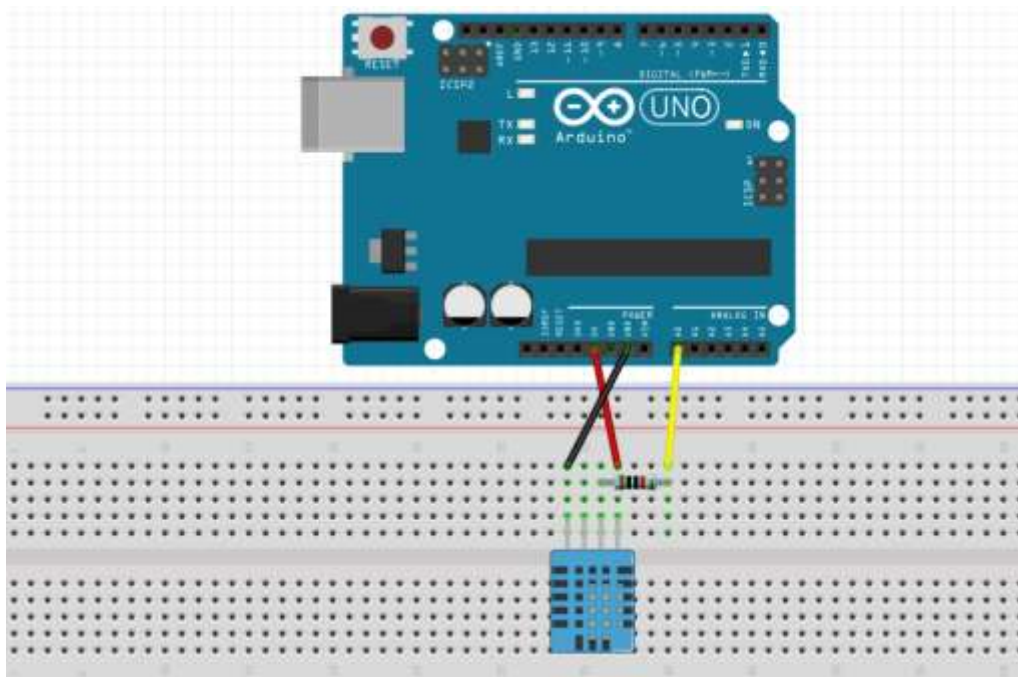


Figura 4.11 – Montagem virtual *DHT11*

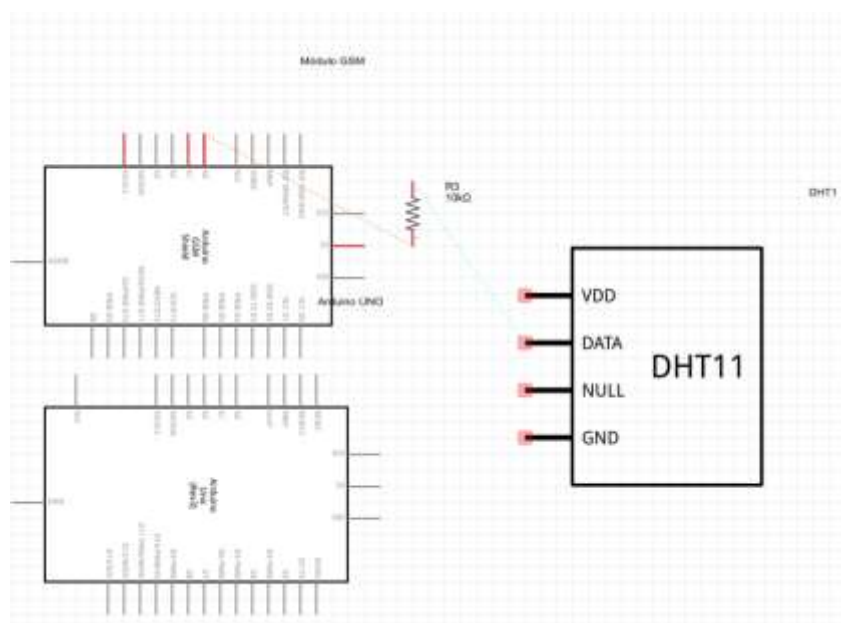


Figura 4.12 – Esquemático *DHT11*

4.6 – Módulo de Relés

O módulo de relés empregado no presente trabalho é alimentado por uma tensão de 5 V e um sinal que aciona abrindo ou fechando o contato. O contato funciona pelo princípio do ímã eletromagnético alimentado por corrente elétrica.

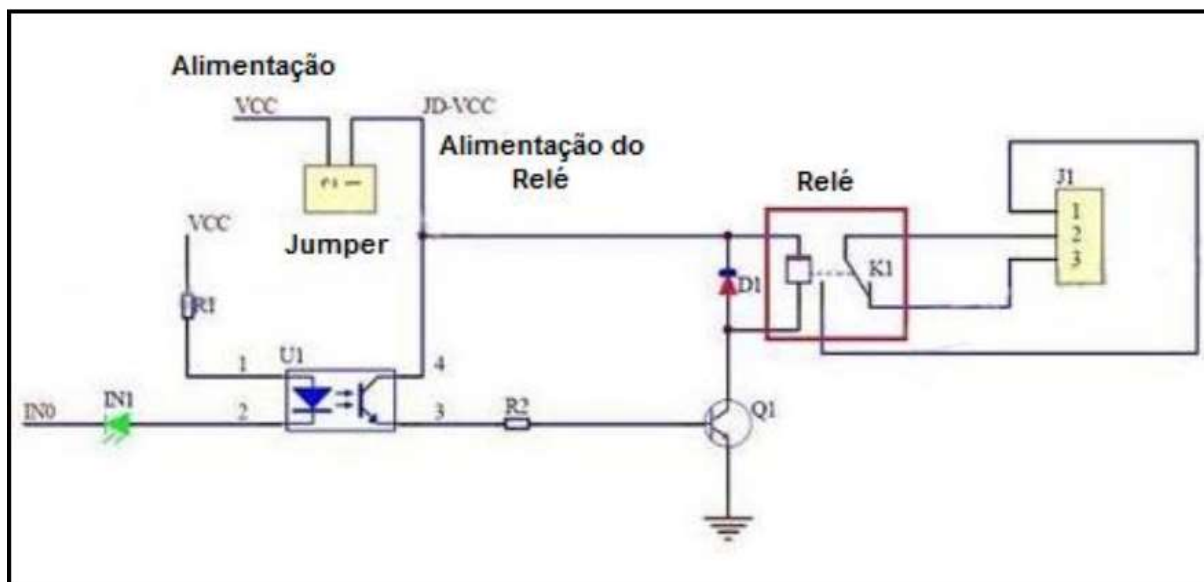


Figura 4.13 – Circuito do relé

A placa com módulos relés usada neste projeto se caracteriza pela associação de um opto-acoplador (U1) – utilizado para proteção das placas Arduino UNO e shield GSM –, um transistor (Q1) que controla o fluxo da corrente no relé, um diodo (D1) e um relé, para cada relé. Apesar de a placa utilizada possuir 4 relés, apenas um é utilizado para o acionamento do umidificador.

CAPÍTULO 5 – IMPLEMENTAÇÃO

Neste capítulo vamos abordar a fase prática deste trabalho em três partes: modelagem, programação e montagem do hardware.

5.1 – Modelagem

Essa modelagem pretende deixar mais intuitivo e compreensível a visualização deste trabalho. Com o auxílio de esquemas montados ao começo do Trabalho e usando o software livre *Lucid Chart*. Essa abordagem a seguir é bastante resumida e abstrai todas as partes do trabalho. A fluxograma a seguir abarca todo o trabalho. As demais partes serão detalhadas à frente.

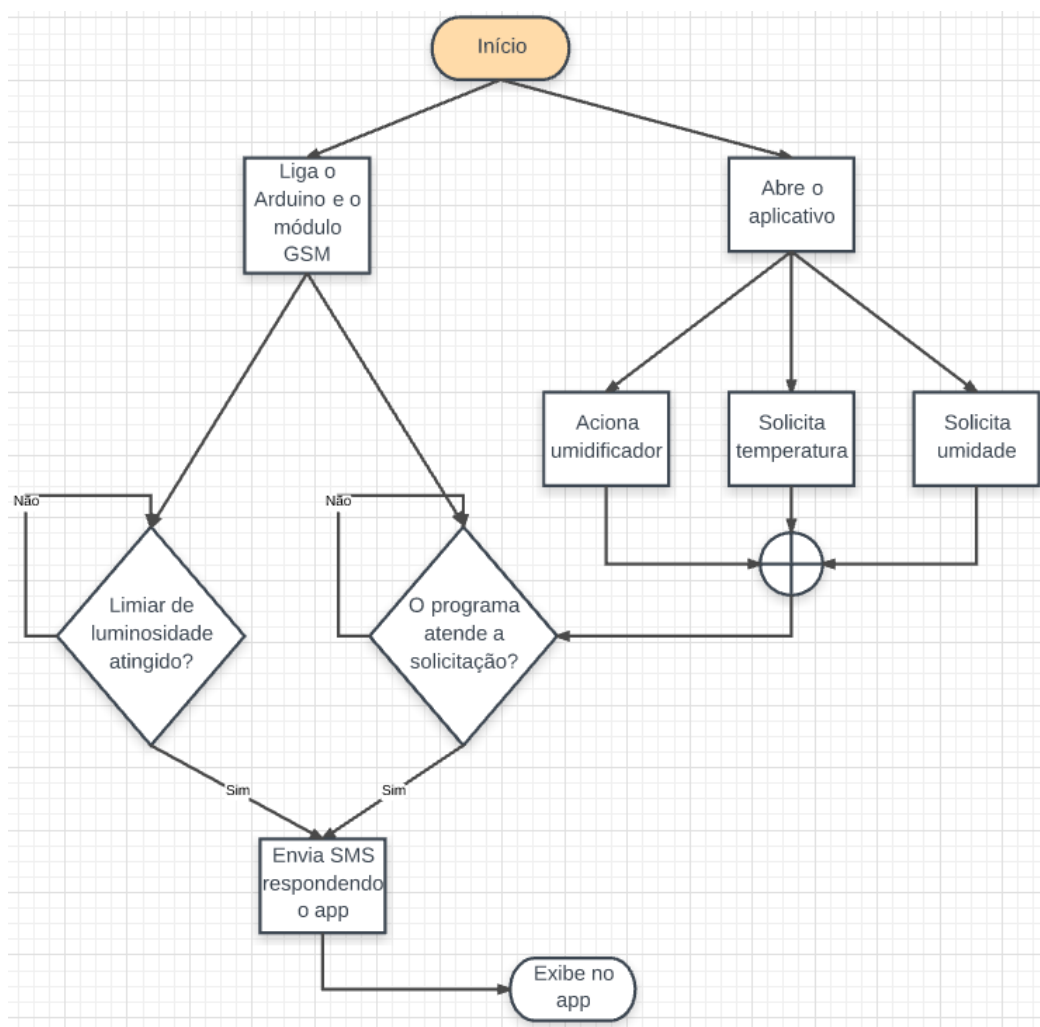


Figura 5.1 – Modelagem do Trabalho completo

A seguir, detalharemos por meio de fluxograma o modo de funcionamento do programa feito para a IDE Arduino. Esta explanação aparenta-se com um pseudocódigo.

Inicialmente, no *setup*, o programa prepara o módulo GSM e o DHT11, inicializando suas sub-rotinas. Em seguida, inicia-se o *loop*.

Nesse loop, o programa procura por mensagens não lidas e caso haja alguma, o programa a responde ao número que enviou a solicitação. Caso não haja mensagem nova, o programa prossegue e realiza a medida das variáveis temperatura, luminosidade e umidade. Caso receba solicitação de acionamento, envia um sinal de 5 V para saída correspondente ao relé do umidificador.

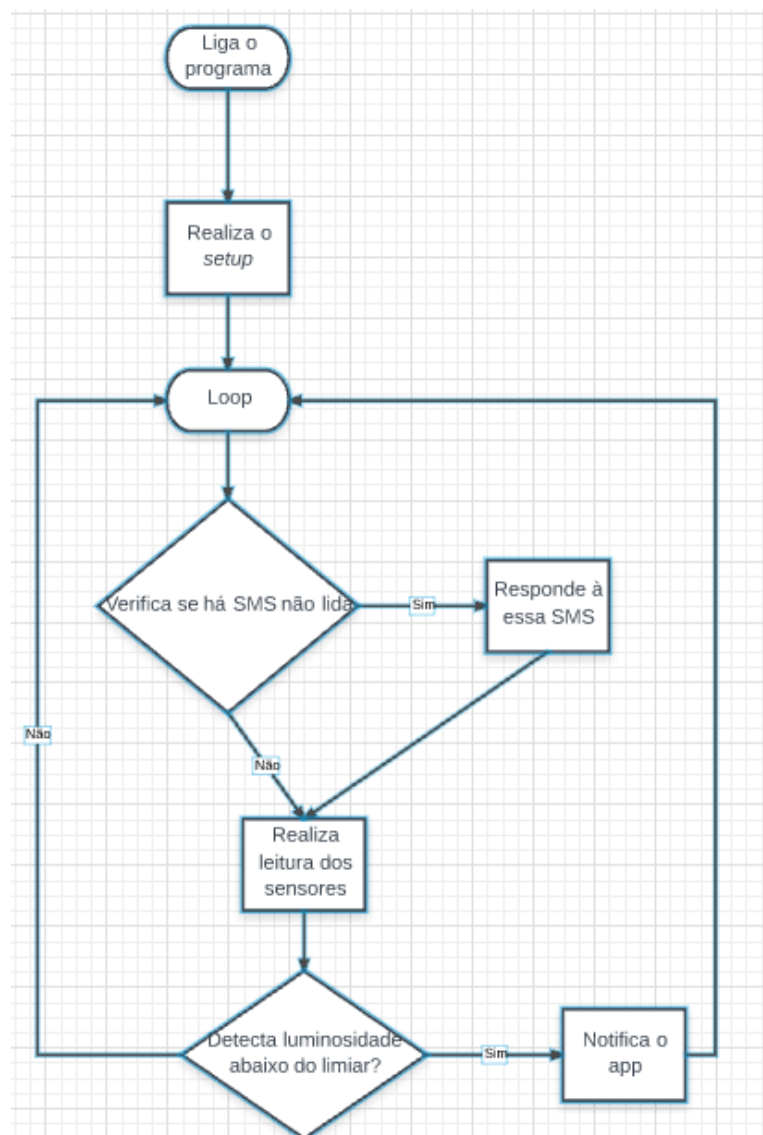


Figura 5.2 – Modelagem programa *IDE Arduino*

Agora apresentaremos a modelagem do programa feito no Android Studio. Ao ser iniciado, o programa abre a única tela que possui. Nessa tela, podemos solicitar temperatura, umidade, acionamento ou desligamento do umidificador. Caso o programa receba notificação de luminosidade, mostra a mensagem de “Presença detectada”.

Assim que recebe a resposta na forma de SMS do módulo GSM, o programa executa uma ação-resposta. Exibe a umidade, temperatura, colore os botões de acionamento e desligamento e caso seja detectado o limiar de luminosidade (presença) mostra mensagem correspondente. A figura a seguir apresenta isso.



Figura 5.3 – Fluxograma *IDE Android Studio*

5.1 – Programação

A partir do fluxograma anteriormente proposto, abordaremos o desenvolvimento do código fonte do Arduino e do Android.

Importante, primeiramente, apontarmos que na fase de testes do Trabalho, foi usada a comunicação serial via computador. Essa fase se fez necessária para detectar erros em cada parte do trabalho. Foram necessários testes na fase de implementação de hardware e a averiguação da correção destes esquemas foi feito a partir do retorno obtido na tela *Serial Monitor* já abordada na introdução teórica da IDE Arduino.

Outro item usado em testes dos programas feitos é o ambiente de simulação da IDE *Android Studio*. Este ambiente emula um aparelho celular com Sistema Operacional Android, e neste aparelho são subidas as programações feitas na IDE do *Android Studio*. A Figura 5.4 a seguir ilustra o ambiente de emulação do Android feita na IDE *Android Studio*.

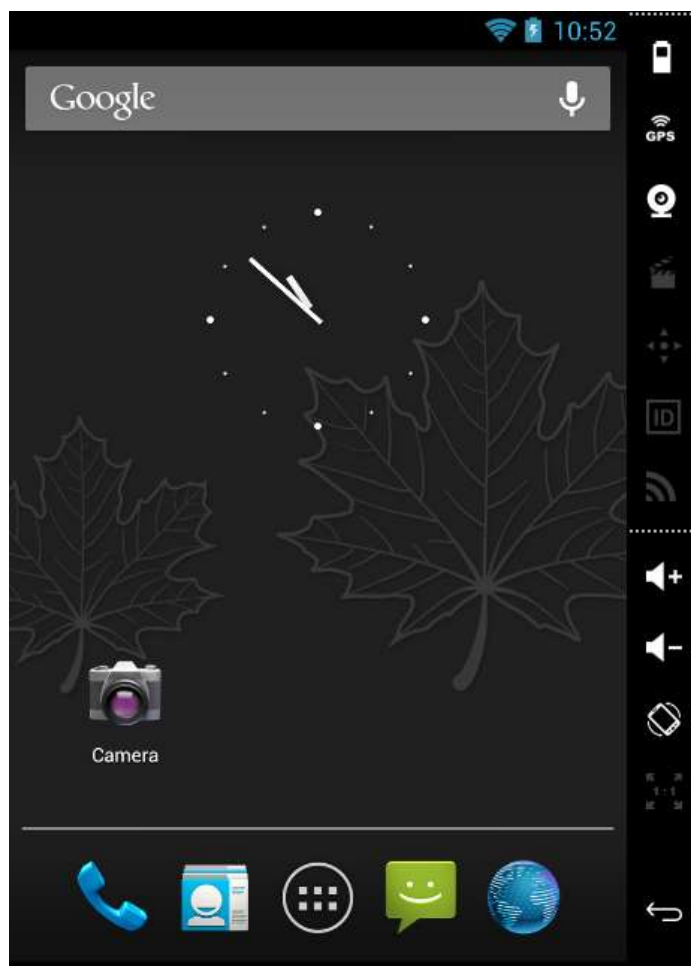
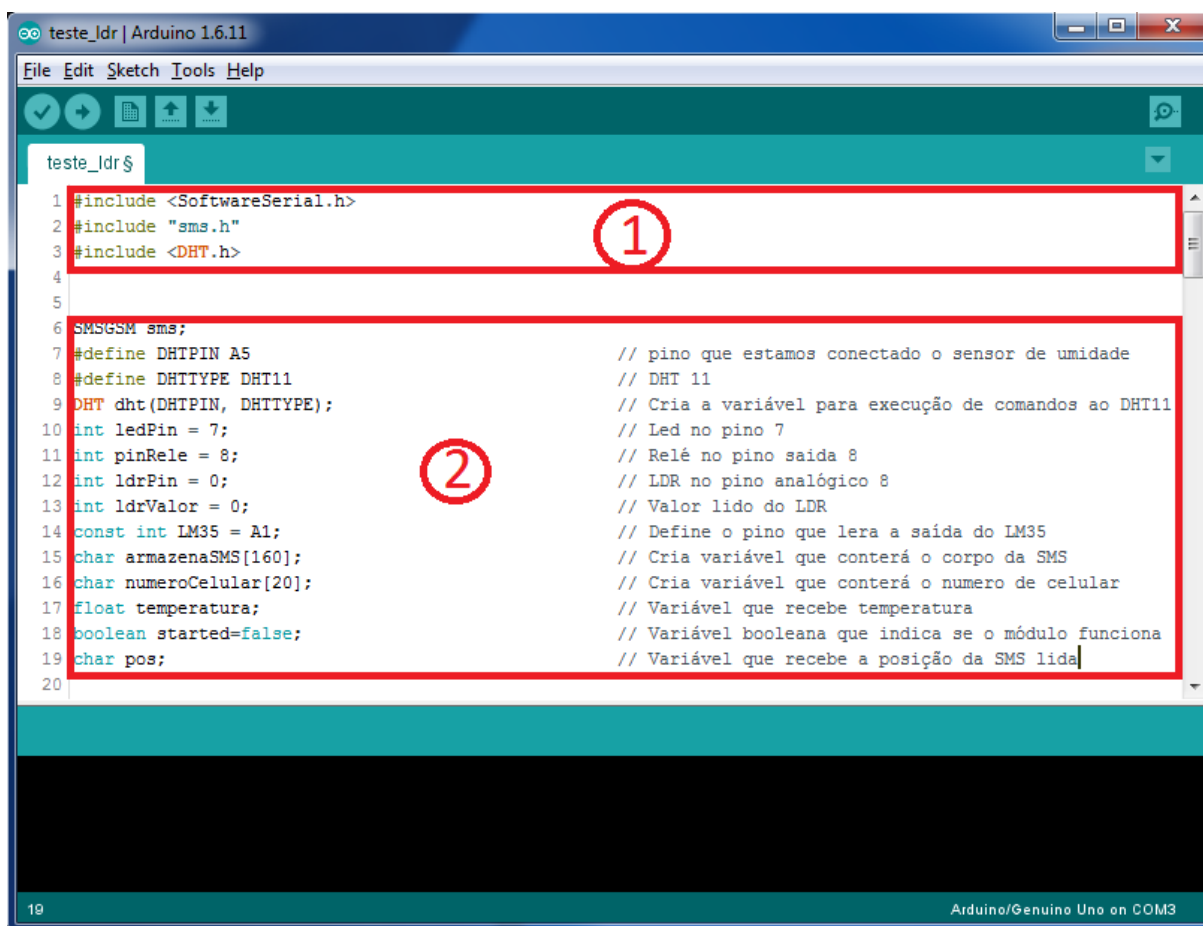


Figura 5.4 – Emulação *IDE Android Studio*

5.1.2 – Programação IDE Arduino

O código foi escrito nas linguagens de programação do Arduino, *Processing* e C. Inicialmente (1), temos a chamada para bibliotecas que serão usadas no programa, as quais já foram apresentadas na introdução teórica, “SoftwareSerial.h”, “sms.h” e “DHT.h”. Em seguida (2) vamos apresentar a preparação de variáveis globais, em geral usada para definir pinos, saídas e entradas e outras variáveis diversas usadas no projeto.

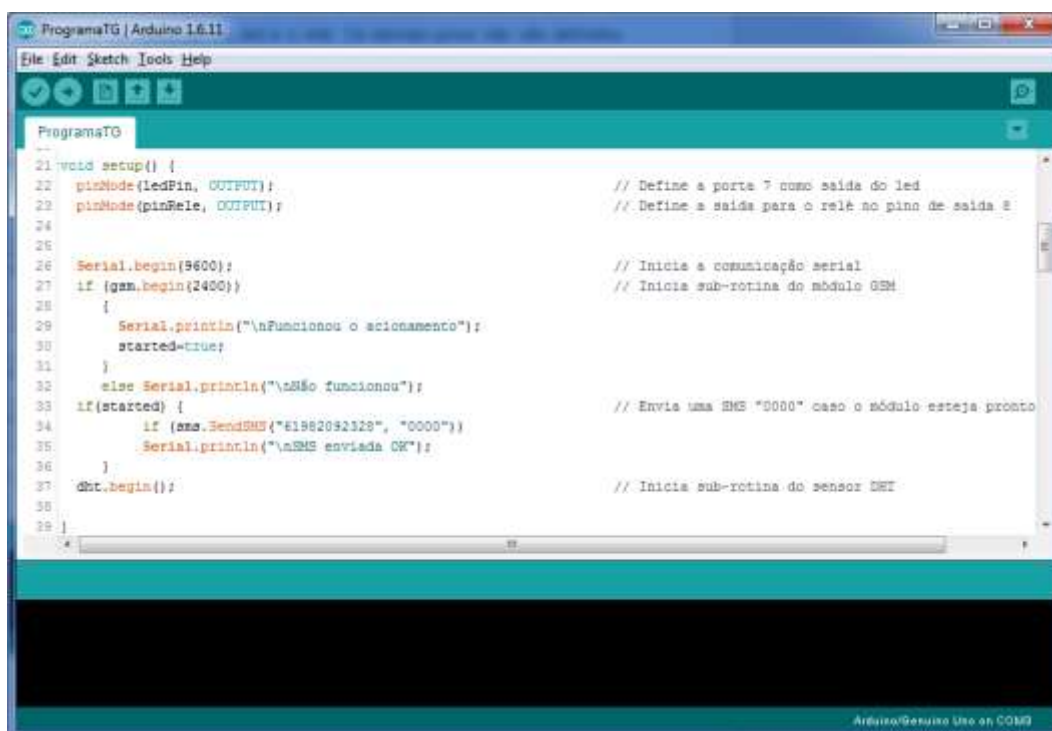


```
teste_ldr | Arduino 1.6.11
File Edit Sketch Tools Help

teste_ldr $
1 #include <SoftwareSerial.h>
2 #include "sms.h"
3 #include <DHT.h>
4
5
6 SMSGSM sms;
7 #define DHTPIN A5 // pino que estamos conectando o sensor de umidade
8 #define DHTTYPE DHT11 // DHT 11
9 DHT dht(DHTPIN, DHTTYPE); // Cria a variável para execução de comandos ao DHT11
10 int ledPin = 7; // Led no pino 7
11 int pinRele = 8; // Relé no pino saída 8
12 int ldrPin = 0; // LDR no pino analógico 8
13 int ldrValor = 0; // Valor lido do LDR
14 const int LM35 = A1; // Define o pino que lera a saída do LM35
15 char armazenaSMS[160]; // Cria variável que conterá o corpo da SMS
16 char numeroCelular[20]; // Cria variável que conterá o numero de celular
17 float temperatura; // Variável que recebe temperatura
18 boolean started=false; // Variável booleana que indica se o módulo funciona
19 char pos; // Variável que recebe a posição da SMS lida
20
```

Figura 5.5 – Início do programa *IDE Arduino*

A seguir, apresentaremos o setup do programa. Nesta parte, iniciamos sub-rotinas que são processadas paralelamente ao programa, iniciar variáveis, definir modos de entrada e saída de pinos. Em nosso programa, são iniciadas as sub-rotinas do módulo GSM e do sensor DHT11. Os pinos de saída para o *led* e o relé. Os demais pinos não são definidos porque são pinos somente de entrada, não necessitam ser definidos.



Figuraa5.6 – Setup IDE Arduino

A seguir, iniciando o *loop*, que é responsável pelo algoritmo do programa em si. Inicialmente são definidas mais variáveis, desta vez locais. A primeira parte do programa recebe uma SMS não lida(nova) do aplicativo Android. Se essa SMS for “1”, o programa aciona o relé do umidificador e retorna a SMS “1”. Caso receba “2” como mensagem, desliga o umidificador e retorna a SMS “2”. Se for o “3”, o programa retorna uma SMS da leitura da temperatura lida no LM35. Recebendo a SMS “5”, o programa retorna a umidade lida no sensor DHT11. Após realizar o procedimento de envio da mensagem, o programa apaga a mensagem recebida para descongestionar a memória do chip SIM GSM, que comporta apenas 20 SMS.



Figura 5.7 – Loop IDE Arduino 1ª parte

A segunda parte do *loop* trata da leitura de umidade feita pelo sensor *DHT11* – o programa retorna uma mensagem de erro caso não seja feita a leitura –, faz leitura de temperatura no sensor *LM35*, realiza a leitura de luminosidade no sensor *LDR* e envia uma mensagem *SMS* “1111” caso o sensor faça uma leitura acima de 72 mW – o que corresponde a uma sombra sobre o sensor –, conta 2000 ms para que o *loop* não seja repetido imediatamente pela *Arduino*. Ao final, ele conta 450 ciclos do *loop* – que correspondem a 15 minutos e acende e apaga um *led*. Este *led* simula a uma lâmpada em uma casa que acende e apaga a cada 15 minutos para simular que há alguém em casa caso os moradores viagem, para evitar ação de estranhos.

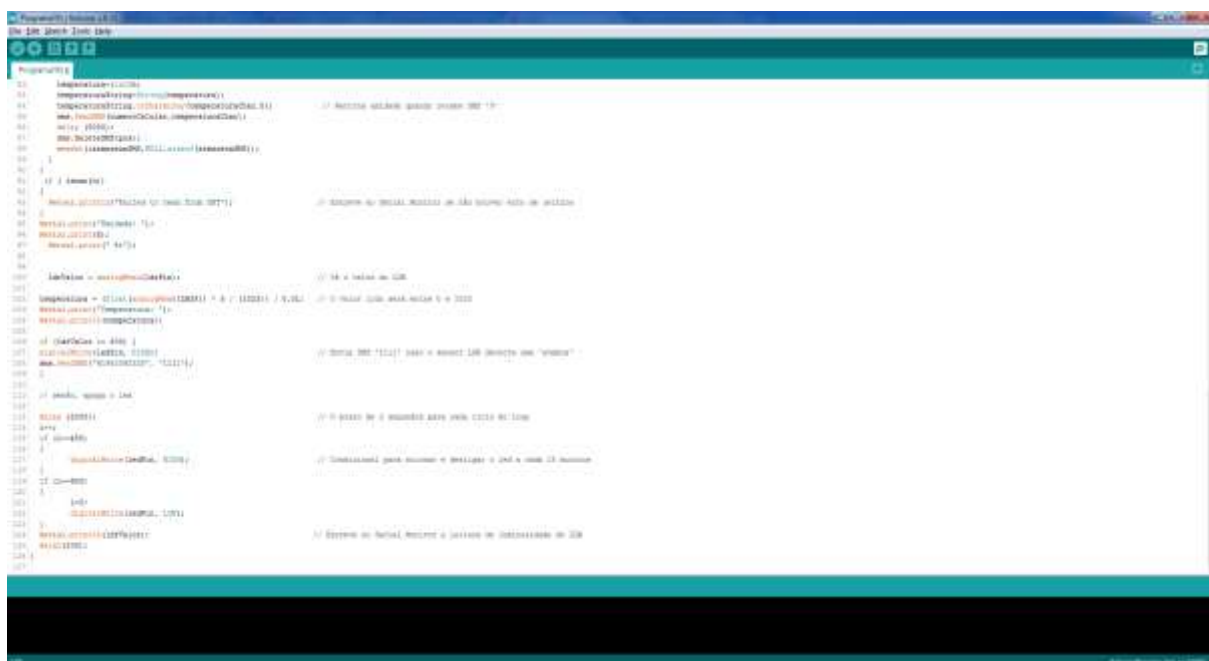


Figura 5.8 – Loop IDE Arduino 2ª parte

5.1.3 – Programação *IDE Android Studio*

A programação feita na *IDE do Android Studio* será dividida em duas partes. A primeira parte trata da chamada de bibliotecas, criação de variáveis e associação de variáveis aos botões e caixas de texto. As bibliotecas usadas neste aplicativo de Android são básicas para criação de interfaces gráficas. São elas: chamada de *Activity*, aparência do aplicativo, responsáveis por envio de *SMS*, criação de botões, caixas de texto e mensagens rápidas na tela.

As variáveis criadas servem para executar funções e procedimentos no algoritmo executado. Após essas criações de variáveis, é criada ao programa para que permaneça em uma única *activity*, criando uma referência para si mesma. Isso porque, durante a confecção deste aplicativo, outras *activities* foram criadas para testar partes diferentes das propostas do aplicativo, como uma *activity* para enviar, uma inicial e outra para receber SMS.

A seguir, é feita a associação de variáveis com os botões e caixas de texto. Abaixo disso, temos a primeira parte do algoritmo em si que chama a função de “sendSMS()” caso seja pressionado os botões no programa.

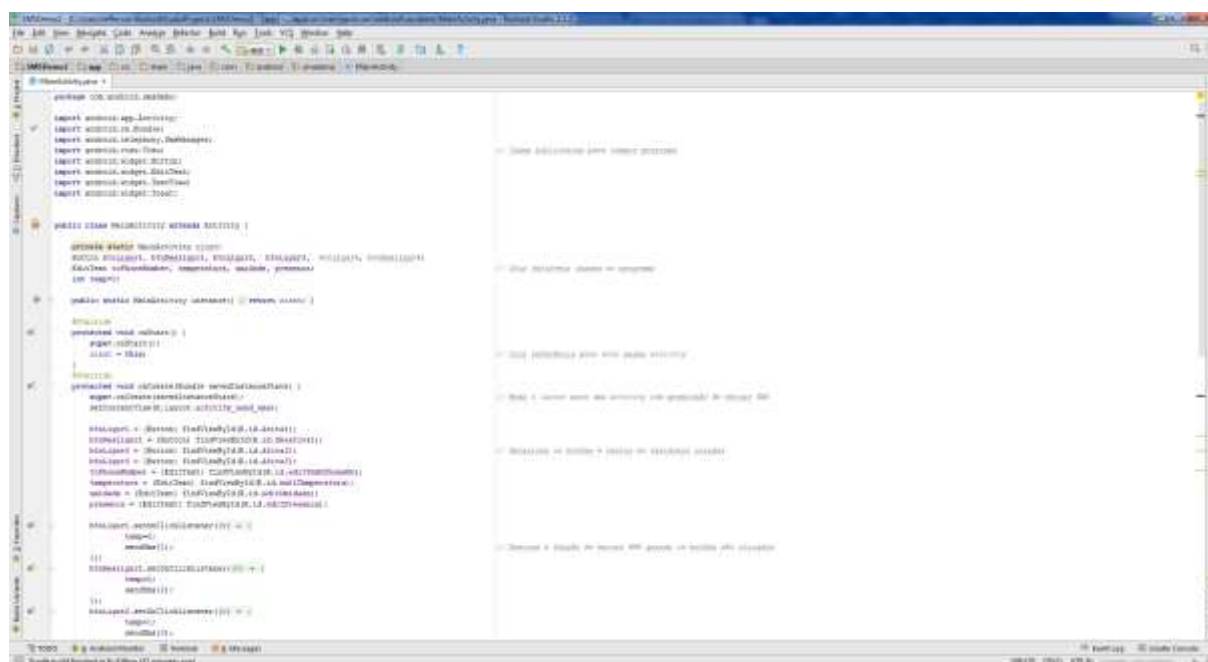


Figura 5.9 – Código-fonte IDE Android Studio 1ª parte

Na segunda parte desse código-fonte, temos as demais funções usadas neste aplicativo Android. Primeiramente, temos uma função que muda a cor dos botões caso o aparelho receba uma SMS do módulo GSM. Também exibe numa caixa de texto editável a temperatura e a umidade medidas pelos sensores já mencionados neste trabalho.

Abaixo, temos o escopo da função que envia SMS que se comunica com o módulo GSM SIM900. As mensagens são o “1”, “2”, “3” e “5”.

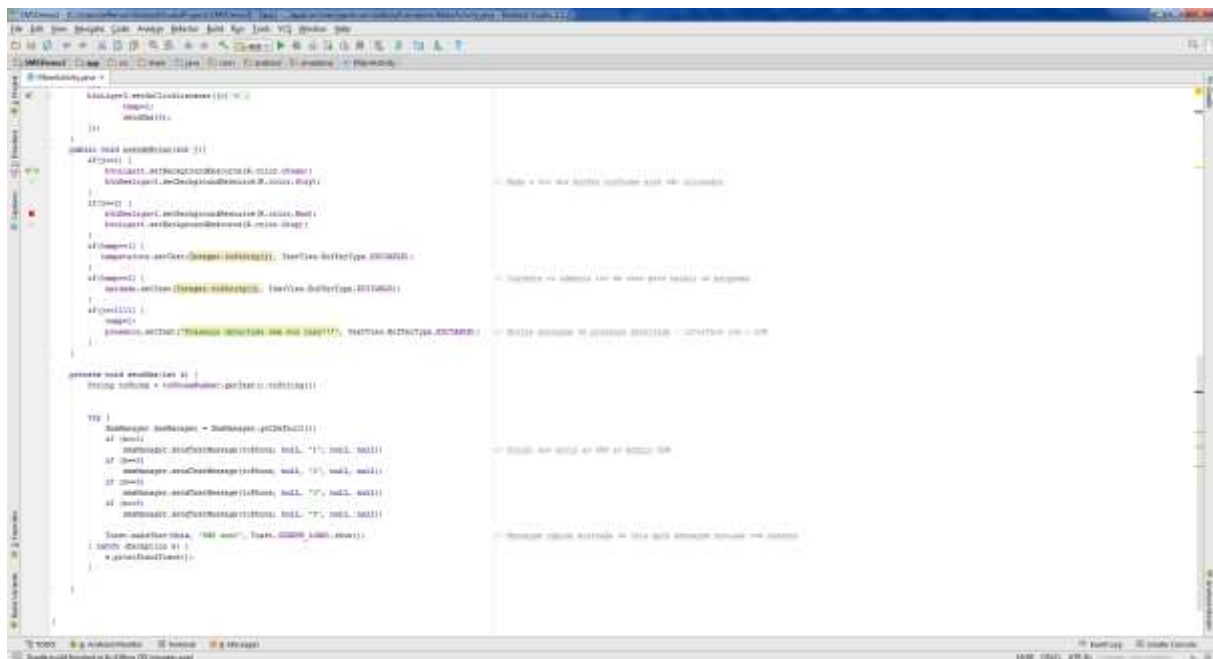


Figura 5.10 – Código-fonte *IDE Android Studio* 2a parte

A Figura 5.11 exibe como fica o layout do programa feito. É importante apontar que a aparência ficou ligeiramente mais alinhada quando subida para o celular.

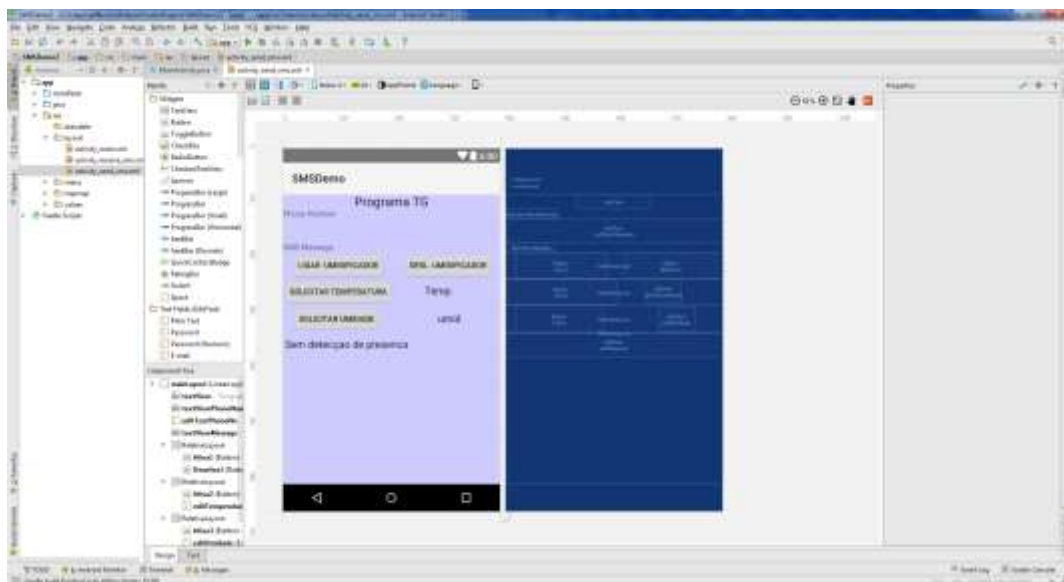


Figura 5.11 – *Layout IDE Android Studio*

5.3 – Montagem do hardware

Os esquemas de ligação dos sensores foram feitos no programa *Fritzing* e apresentados na introdução teórica. A seguir, apresentaremos a montagem virtual de todos os sensores ligados ao Arduino e ao módulo de relés, bem como o módulo de relés ligado ao umidificador e à rede 220 V.

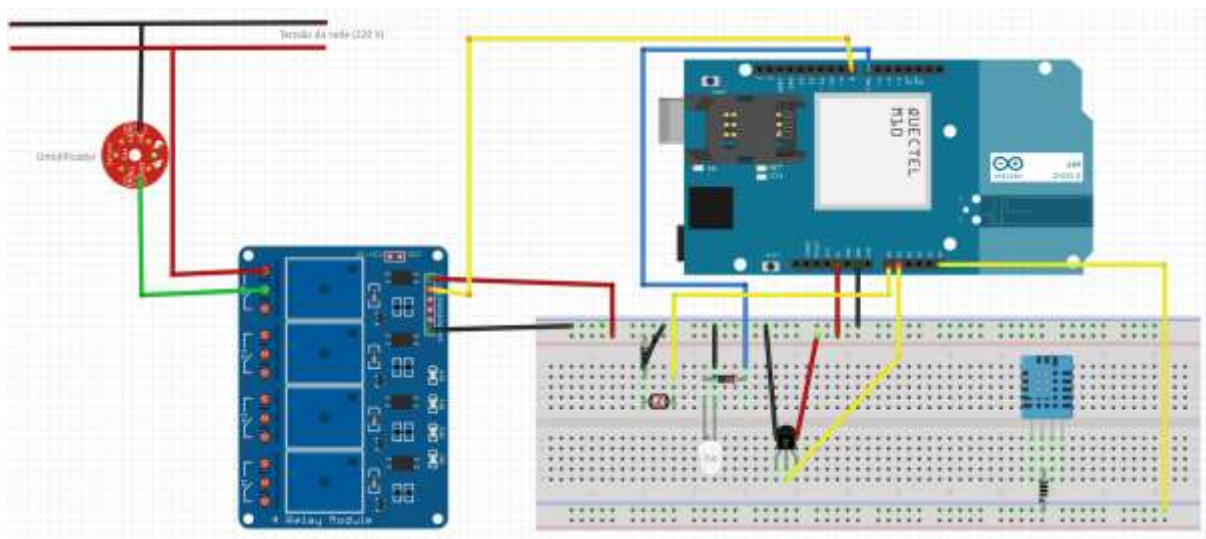


Figura 5.12 – Montagem virtual do *hardware*

Em seguida, o esquemático também feito no *Fritzing*.

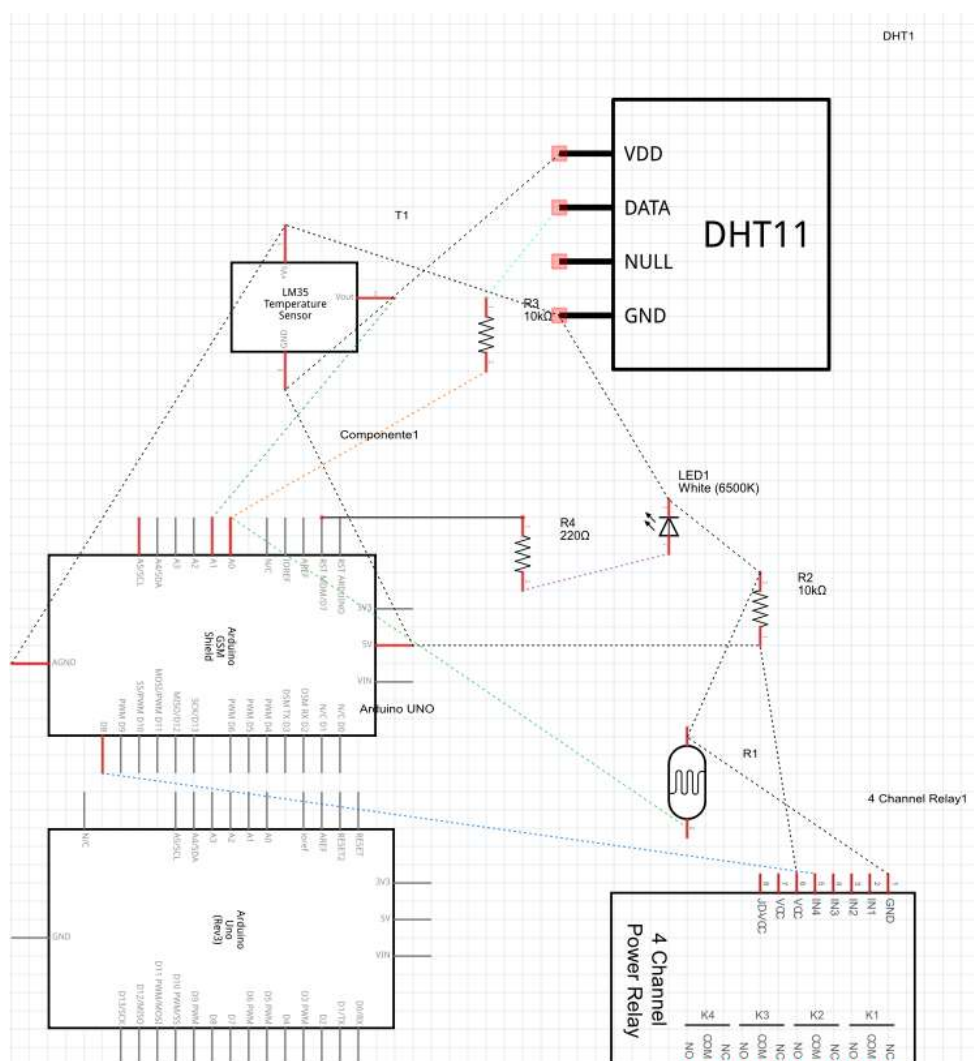


Figura 5.13 – Esquemático geral

CAPÍTULO 6 – RESULTADOS OBTIDOS

Os resultados obtidos agora serão expostos conforme fomos testando cada parte do trabalho. A começar pela parte programada na IDE do Arduino. A primeira figura (6.1) exibe o *setup* do módulo GSM. Os demais *setups* terão sua comprovação adiante.

6.1 – Resultados de *software*

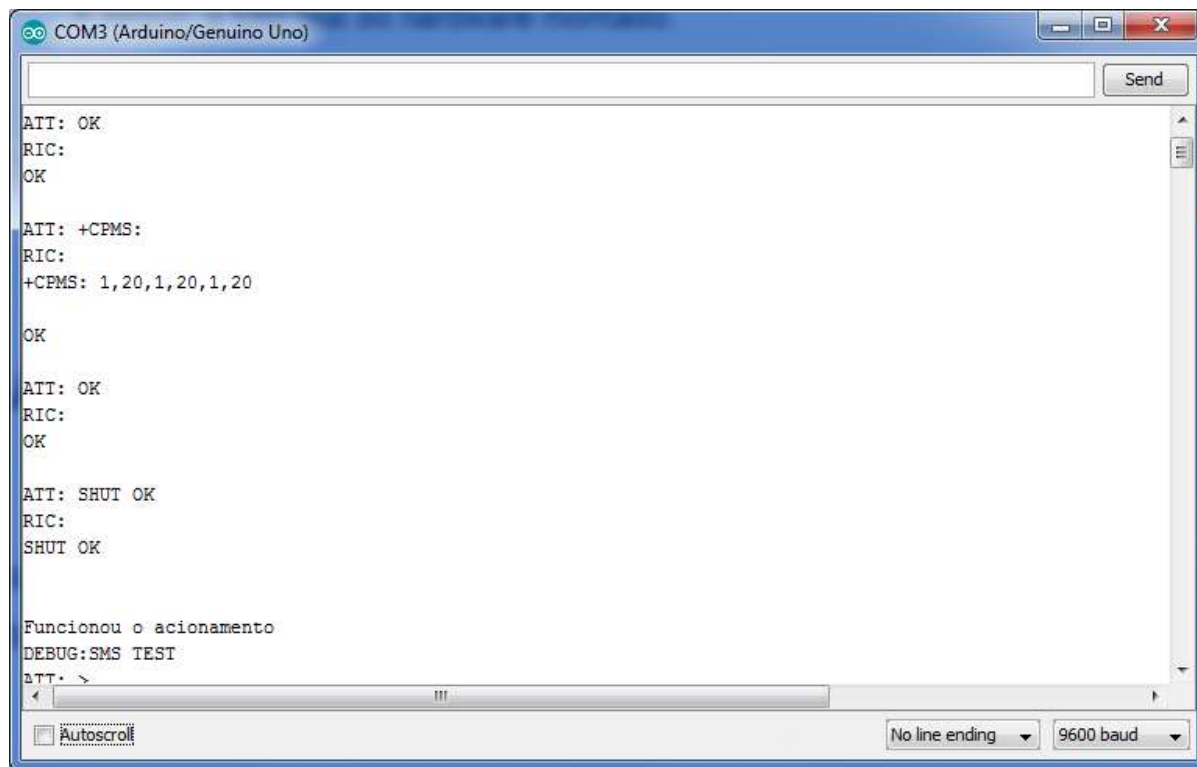


Figura 6.1 – Setup do módulo GSM (*Serial Monitor*)



Figura 6.2 – Mensagem recebida ao iniciar o módulo

A Figura 6.3 a seguir ilustra o retorno das medidas nos sensores DHT (umidade), LM35 (temperatura) e LDR (luminosidade).

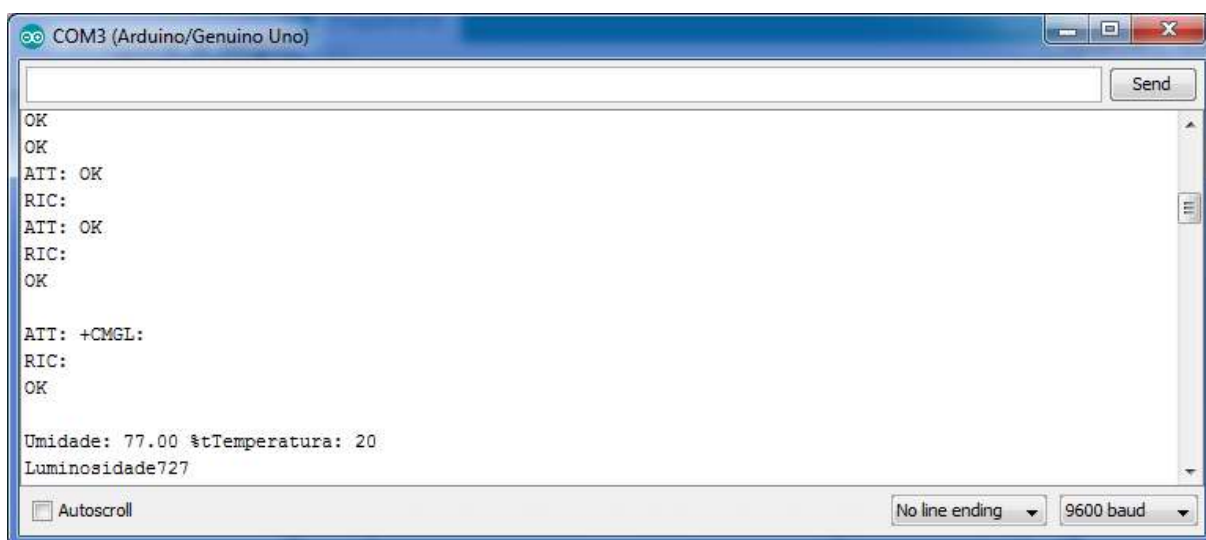


Figura 6.3 – Medidas dos sensores (*Serial Monitor*)

Esta Figura 6.4 a seguir ilustra o que o *Serial Monitor* exibe quando enviamos as mensagens “1”, “2”, “3” e “5”.

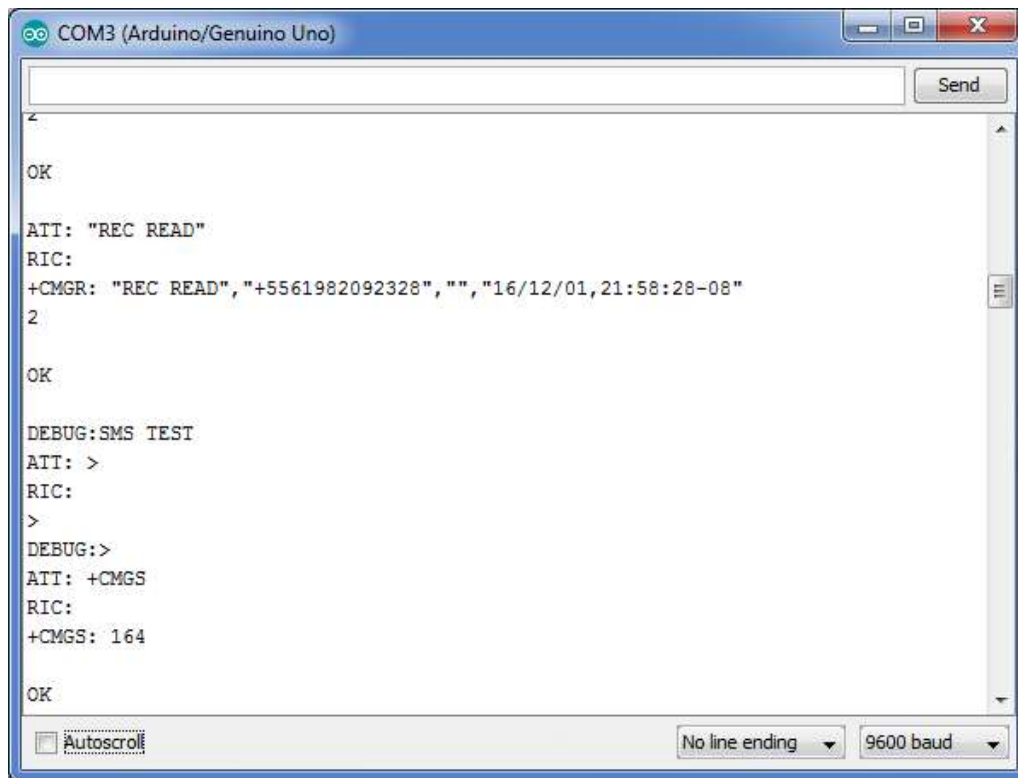


Figura 6.4 – Serial Monitor respondendo SMS

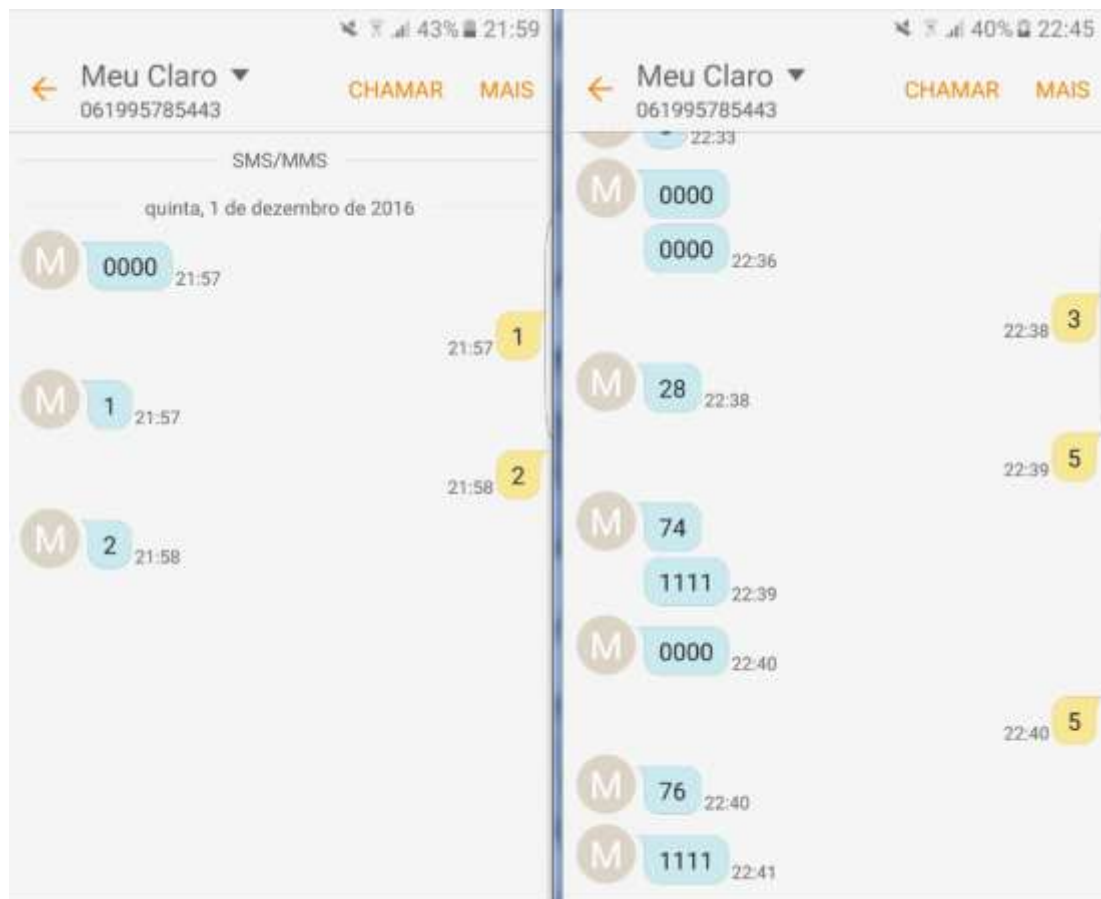


Figura 6.5 – Celular enviando e recebendo SMS ao módulo GSM

A Figura 6.5 mostra a mensagem enviada e recebida pelo celular que porta o aplicativo. Observe que o programa responde com “28” quando enviamos o número “3”, ou seja, envia como resposta a temperatura. A resposta ao “5” é “70”, correspondente à umidade no presente momento da medição. Além de enviar “1111” quando é colocada algum objeto obstruindo a luz incidente no LDR.

As figuras a seguir (Figuras 6.6 e 6.7) se referem aos resultados obtidos da programação feita na IDE Android Studio. A primeira se refere ao ligar e desligar do umidificador. Após apertar o botão e esperar alguns segundos (o tempo de enviar a SMS, o módulo receber e responder com outra SMS que aciona o programa, correspondente à Figura 6.5 acima).

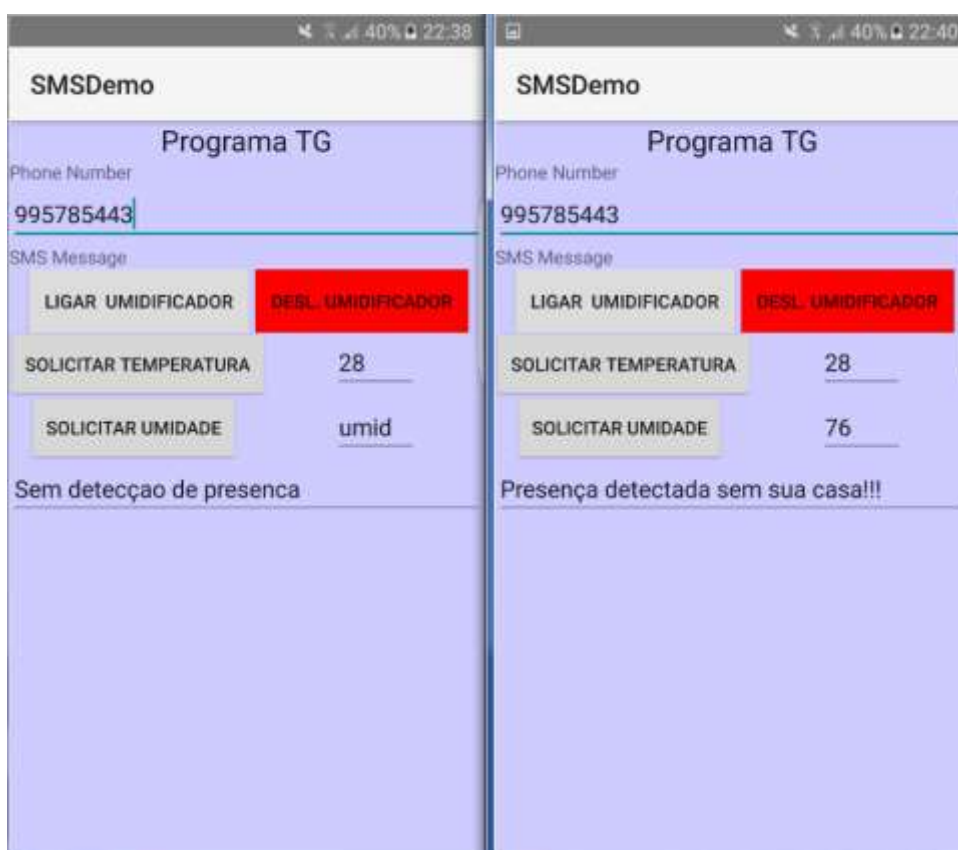


Figura 6.6 – Testes feitos no app 1ª parte



Figura 6.7 – Teste feitos no *app* 2ª parte

A Figura 6.7 se refere a solicitações de temperatura e umidade, além de apresentar a mensagem de presença detectada após obstruir a luz incidente no LDR.

6.2 – Resultados de *hardware*

A Figura 6.8 apresenta o *hardware* relé e umidificador montado.



Figura 6.8 – Montagem do relé com o umidificador



Figura 6.9 – Módulo GSM e Arduino

As Figuras 6.9 e 6.10 apresentam o módulo e a *protoboard* montadas e prontas para funcionamento.

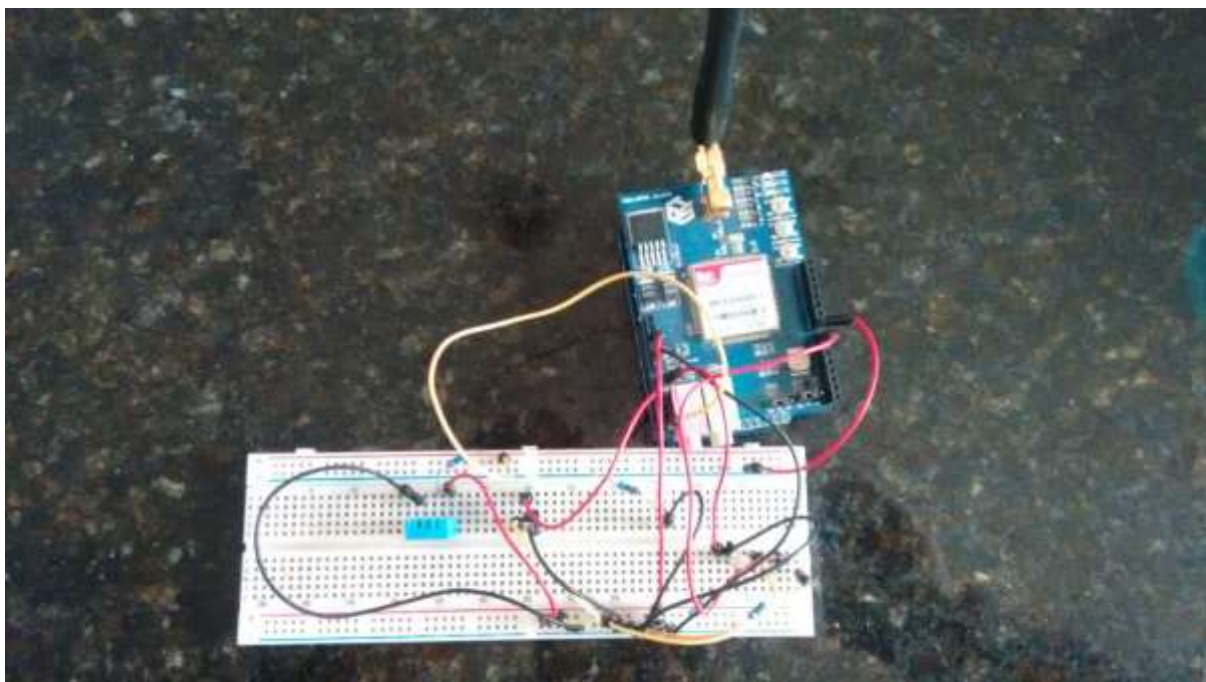


Figura 6.10 – Módulo GSM, Arduino e sensores na *protoboard*

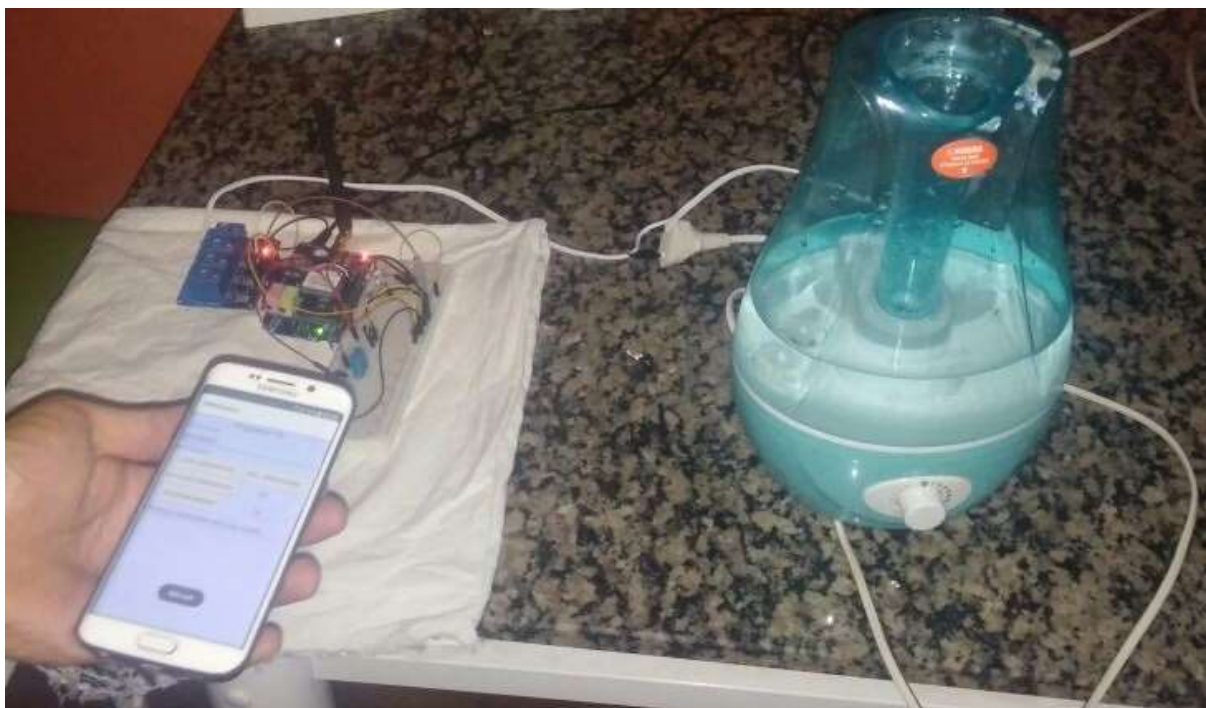


Figura 6.11 – Trabalho montado pronto para uso

A Figura 6.11 apresenta o projeto agora pronto para execução, já foi solicitado ao módulo a temperatura, umidade e foi registrada mudança na luminosidade, conforme cobriu-se o LDR. As figuras 6.12 e 6.13 apresentam o umidificador sendo acionado e desligado, respectivamente.



Figura 6.12 – Esquema montado e acionado



Figura 6.13 – Esquema montado e desligado

6.3 – Problemas encontrados

O desenvolvimento deste trabalho foram encontradas pequenas dificuldades referentes aos *softwares*, pois a diversidade de atualizações, bibliotecas e pequenas peculiaridades na elaboração e funcionamento. Contudo, a vasta documentação presente na internet e várias tentativas supriram essa barreira.

Alguns atrasos no envio de *SMS* podem eventualmente prejudicar o funcionamento do aplicativo para Android. É preciso, pois, ter paciência quando se envia um comando pelo aplicativo (o prazo de enviar e receber uma *SMS*). Significa dizer que o funcionamento deste trabalho depende do tempo que uma *SMS* leva para ser enviada e recebida em uma dada área de cobertura *GSM*.

Particularidades como a confidencialidade de quem envia e recebe a mensagem não foram tratadas nesse trabalho.

CAPÍTULO 7 – CONSIDERAÇÕES FINAIS

7.1 – Conclusões

Este trabalho se propôs a criar uma interface de automação remota residencial. Embora de pequena complexidade, o arranjo exposto conseguiu alcançar o objetivo. Além disso, todas as partes interligadas (relé, módulo *GSM*, *Arduino*, aplicativo *Android*, gama de sensores) são facilmente expansíveis. Significa dizer que o aplicativo *Android* pode aumentar o comando de variáveis, pode-se agregar mais sensores, o relé pode acionar mais eletrodomésticos.

7.2 – Sugestões para trabalhos futuros

A demanda por automação residencial é uma boa aposta para o futuro. Não apenas residencial, mas automações simples tornam a vida do engenheiro com carreira relacionada a tecnologias eletrônicas mais fácil

Algumas sugestões para aprofundar este trabalho:

- criar um protótipo mais barato, usando outras tecnologias que enviam *SMS*;
- agregar mais instrumentos a este trabalho, com o objetivo de estender as funcionalidades ou melhorar as existentes;
- incluir câmera para monitoramento doméstico.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] MURATORI, José Roberto; DAL BÓ, Paulo Henrique. Automação Residencial. Automação residencial: histórico, definições e conceitos, São Paulo, mar./jul. 2011. 70 p.

[2] ROVERI, Michael Rubens. Automação Residencial, 2012. Trabalho de Conclusão de Curso (Curso Superior de Tecnologia em Redes de Computadores)- Faculdade POLITEC, Santa Bárbara d'Oeste, 2012.

[3] MAIA, Gustavo Moura Fé. Acionamento Remoto De Portões Elétricos Via Celular Através De Microcontrolador, 2012. Trabalho de Conclusão de Curso (Graduação em Engenharia da Computação)- Faculdade de Tecnologia e Ciências Sociais Aplicadas - FATECS, Centro Universitário de Brasília - UniCEUB, Brasília DF, 2012, 17p.

[4] SIQUEIRA, Charles de Souza; BOAS, Pablo Pinheiro Batista Villas. Projeto De Automação Residencial Utilizando Um Microcontrolador Da Família 8051 E Supervisionado Por Uma Plataforma Desenvolvida No Eclipse E3, 2011. Trabalho de Conclusão de Curso (Graduação em Engenharia Elétrica)- Escola de Engenharia Elétrica e de Computação, Universidade Federal de Goiás, Goiânia GO, 2011, 17p.

[5] ARDUINO GSM SHIELD. Open Eletronics, Open source eletronic projects. Disponível em: <<https://www.arduino.cc/>> Acesso: 03 de dezembro de 2016.

[6] ARDUINO UNO, Arduino UNO Front. Arduino UNO Board. Disponível em: <<https://www.arduino.cc/>>: Acesso: 03 de dezembro de 2016.

[7] DHT11, MONITORANDO TEMPERATURA E UMIDADE COM O SENSOR DHT11, Disponível em: <<http://blog.filipeflop.com/sensores/monitorando-temperatura-e-umidade-com-o-sensor-dht11.html>>. Acesso: 03 de dezembro de 2016.

[8] LM35, LM35 – Medindo temperatura com Arduino. Disponível em: <<http://blog.vidadesilicio.com.br/arduino/basico/lm35-medindo-temperatura-com-arduino/>>. Acesso: 03 de dezembro de 2016.

[9] Android Studio, Perguntas relacionadas. Disponível em:
<<http://pt.stackoverflow.com/questions/tagged/android-studio>> Acesso: 03 de dezembro de 2016.

[10] Relé 5 V 4 canais, CONTROLANDO LÂMPADAS COM MÓDULO RELÉ ARDUINO, Disponível em: <<http://blog.filipeflop.com/modulos/controla-modulo-rele-arduino.html>> Acesso em: 03 de dezembro de 2016.

[11] Sending/Receiving SMS via Arduino. Disponível em:
<<http://stackoverflow.com/questions/15658633/sending-receiving-sms-via-arduino>> Acesso: 03 de dezembro de 2016

APÊNDICE A – CÓDIGO FONTE IDE ARDUINO

```
//Este código está comentado

#include <SoftwareSerial.h>

#include "sms.h"

#include <DHT.h>


MSGSMS sms;                                // Cria variável que chama as funções SMS

#define DHTPIN A5                          // pino que estamos conectado o sensor de
umidade

#define DHTTYPE DHT11                      // DHT 11

DHT dht(DHTPIN, DHTTYPE);                 // Cria a variável para execução de
comandos ao DHT11

int ledPin = 8;                            // Led no pino 8

int pinRele = 7;                          // Relé no pino saída 7

int ldrPin = 0;                           // LDR no pino analógico 8

int ldrValor = 0;                         // Valor lido do LDR

const int LM35 = A1;                      // Define o pino que lera a saída do LM35

char armazenaSMS[160];                    // Cria variável que conterà o corpo da
SMS

char numeroCelular[20];                   // Cria variável que conterà o numero de
celular

float temperatura;                        // Variável que recebe temperatura

boolean started=false;                    // Variável booleana que indica se
o módulo funciona

char pos;                                // Variável que recebe a posição da
SMS lida


void setup() {

    pinMode(ledPin, OUTPUT);              // Define a porta 7 como saída
do led
```

```

        pinMode(pinRele, OUTPUT);                                // Define a saída para o relé
no pino de saída 8

Serial.begin(9600);                                              // Inicia a comunicação serial
if (gsm.begin(2400))                                            // Inicia sub-rotina do módulo GSM
{
    Serial.println("\nFuncionou o acionamento");
    started=true;
}
else Serial.println("\nNão funcionou");

if(started) {                                                  // Envia uma SMS "0000" caso o módulo
esteja pronto
    if (sms.SendSMS("61982092328", "0000"))
        Serial.println("\nSMS enviada OK");
}

dht.begin();                                                  // Inicia sub-rotina do sensor DHT

}

void loop() {

    float h = dht.readHumidity();                                // Variável 'h' recebe indicação de
umidade

    float t;

    int temperatura;

    int i=0;

    char temperaturaChar[5];                                    // Inicia diversas variáveis

    String temperaturaString;

    temperatura = (float(analogRead(LM35)) * 5 / (1023)) / 0.01;

```

```

t=(int)temperatura;

pos = sms.IsSMSPresent(SMS_UNREAD);           // Salva a posição de
uma SMS não lida na variável pos

if(pos){

    sms.GetSMS(pos, numeroCelular, 20, armazenaSMS, 160);    // Função que
recebe uma mensagem não lida

    if(!strcmp(armazenaSMS,"1"))

    {

        Serial.print(armazenaSMS);

        digitalWrite(7, HIGH);

        if(sms.SendSMS(numeroCelular,"1"))                // Caso receba uma SMS
com conteúdo '1', liga o umidificador

        delay (5000);                                     // e retorna outra SMS '1' notificando
quem solicitou.

        sms.DeleteSMS(pos);                               // Apaga a SMS recebida. As 4
funções abaixo são análogas

        memset(&armazenaSMS,NULL,sizeof(armazenaSMS));

    }

    if(!strcmp(armazenaSMS,"2"))

    {

        digitalWrite(7, LOW);

        sms.SendSMS(numeroCelular,"2");                   // Analogamente à anterior,
porém desliga com o número '2'

        delay (5000);

        sms.DeleteSMS(pos);

        memset(&armazenaSMS,NULL,sizeof(armazenaSMS));

    }

    if(!strcmp(armazenaSMS,"3"))

    {

        temperatura=(int)t;

        temperaturaString=String(temperatura);           // Retorna a temperatura
quando recebe SMS '3'

```

```

    temperaturaString.toCharArray(temperaturaChar,5);

    sms.SendSMS(numeroCelular,temperaturaChar);

    delay (5000);

    sms.DeleteSMS(pos);

    memset(&armazenaSMS,NULL,sizeof(armazenaSMS));
}

if(!strcmp(armazenaSMS,"5"))
{
    temperatura=(int)h;

    temperaturaString=String(temperatura);

    temperaturaString.toCharArray(temperaturaChar,5);    // Retorna umidade
quando recebe SMS '5'

    sms.SendSMS(numeroCelular,temperaturaChar);

    delay (5000);

    sms.DeleteSMS(pos);

    memset(&armazenaSMS,NULL,sizeof(armazenaSMS));
}
}

if ( isnan(h))
{
    Serial.println("Failed to read from DHT");    // Escreve no Serial Monitor se
não houver erro de leitura
}

Serial.print("Umidade: ");

Serial.print(h);

Serial.print(" %t");

    ldrValor = analogRead(ldrPin);    // Lê o valor do LDR

```

```

    temperatura = (float(analogRead(LM35)) * 5 / (1023)) / 0.01; // O valor lido será
entre 0 e 1023

    Serial.print("Temperatura: ");

    Serial.println(temperatura);


    if (ldrValor >= 800) {

        sms.SendSMS("61982092328", "1111"); // Envia SMS '1111' caso
o sensor LDR detecte uma 'sombra'

    }


    // senão, apaga o led


    delay (2000); // O prazo de 2 segundos para cada
ciclo do loop

    i++;

    if (i==450)

    {

        digitalWrite(8, HIGH); // Condicional para acionar e desligar o
led a cada 15 minutos

    }

    if (i==900)

    {

        i=0;

        digitalWrite(8, LOW);

    }

    Serial.print("Luminosidade");

    Serial.println(ldrValor); // Escreve no Serial Monitor a leitura
de luminosidade do LDR

    delay(100);

}

```

APÊNDICE B – CÓDIGO FONTE IDE ANDROID STUDIO

```
// Código JAVA

package com.android.smsdemo;

import android.app.Activity;

import android.os.Bundle;

import android.telephony.SmsManager;

import android.view.View; // Chama
bibliotecas para compor programa

import android.widget.Button;

import android.widget.EditText;

import android.widget.TextView;

import android.widget.Toast;

public class MainActivity extends Activity {

    private static MainActivity ninst;

    Button btnLigar1, btnDesligar1, btnLigar2, btnLigar3;

    EditText toPhoneNumber, temperatura, umidade, presenca;
// Cria variáveis usadas no programa

    int temp=0;

    public static MainActivity instance() {

        return ninst;

    }

    @Override

    protected void onStart() {

        super.onStart();

        ninst = this; // Cria referência
para esta mesma activity

    }

    @Override

    protected void onCreate(Bundle savedInstanceState) {
```

```

        super.onCreate(savedInstanceState); //
Muda o layout para uma activity com permissão de enviar SMS

        setContentView(R.layout.activity_send_sms);

        btnLigar1 = (Button) findViewById(R.id.Ativa1);

        btnDesligar1 = (Button) findViewById(R.id.Desativa1);

        btnLigar2 = (Button) findViewById(R.id.Ativa2); //
Relaciona os botões e textos às variáveis criadas

        btnLigar3 = (Button) findViewById(R.id.Ativa3);

        toPhoneNumber = (EditText) findViewById(R.id.editTextPhoneNo);

        temperatura = (EditText) findViewById(R.id.editTemperatura);

        umidade = (EditText) findViewById(R.id.editUmidade);

        presenca = (EditText) findViewById(R.id.editPresenca);

        btnLigar1.setOnClickListener(new View.OnClickListener() {

            @Override

            public void onClick(View v) {

                temp=0;

                sendSms(1); // Executa a
função de enviar SMS quando os botões são clicados

            }

        });

        btnDesligar1.setOnClickListener(new View.OnClickListener() {

            @Override

            public void onClick(View v) {

                temp=0;

                sendSms(2);

            }

        });

        btnLigar2.setOnClickListener(new View.OnClickListener() {

            @Override

            public void onClick(View v) {

                temp=1;

```



```

        sendSms(3);
    }
});

btnLigar3.setOnClickListener(new View.OnClickListener() {

    @Override

    public void onClick(View v) {

        temp=2;

        sendSms(5);

    }

});

}

public void acendeBotao(int j){

    if(j==1111) {

        temp=0;

        presença.setText("Presença detectada sem sua casa!!!",
TextView.BufferType.EDITABLE); // Mostra mensagem de presença detectada - interface
com o LDR

    }

    if(j==1) {

        btnLigar1.setBackgroundResource(R.color.Green);

        btnDesligar1.setBackgroundResource(R.color.Gray);
// Muda a cor dos botões conforme eles são acionados

    }

    if(j==2) {

        btnDesligar1.setBackgroundResource(R.color.Red);

        btnLigar1.setBackgroundResource(R.color.Gray);

    }

    if(temp==1) {

        temperatura.setText(Integer.toString(j), TextView.BufferType.EDITABLE);

    }

}

```

```

        if(temp==2) {
            // Converte os
            números int em char para exibir no programa

            unidade.setText(Integer.toString(j), TextView.BufferType.EDITABLE);
        }
    }

    private void sendSms(int k) {
        String toPhone = toPhoneNumber.getText().toString();

        try {
            SmsManager smsManager = SmsManager.getDefault();

            if (k==1)

                smsManager.sendTextMessage(toPhone, null, "1", null, null);
// Função que envia as SMS ao módulo GSM

            if (k==2)

                smsManager.sendTextMessage(toPhone, null, "2", null, null);

            if (k==3)

                smsManager.sendTextMessage(toPhone, null, "3", null, null);

            if (k==5)

                smsManager.sendTextMessage(toPhone, null, "5", null, null);

            Toast.makeText(this, "SMS sent", Toast.LENGTH_LONG).show();
// Mensagem rápida mostrada na tela após mensagem enviada com sucesso

        } catch (Exception e) {

            e.printStackTrace();

        }

    }
}

```

Código Layout

```

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

    android:layout_width="match_parent"

```

```

    android:layout_height="match_parent"
    android:orientation="vertical"
    android:id="@+id/mainLayout"
    android:weightSum="1"
    android:background="#ffcecbff" >
<TextView
    android:id="@+id/textView"
    android:text="Programa TG"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:layout_gravity="center_horizontal"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
<TextView
    android:id="@+id/textViewPhoneNumber"
    android:text="Phone Number"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
<EditText
    android:id="@+id/editTextPhoneNo"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:inputType="phone" />
<TextView
    android:id="@+id/textViewMessage"
    android:text="SMS Message"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
<RelativeLayout
    android:layout_width="fill_parent"
    android:layout_height="wrap_content">

```

<Button

```
    android:text="Ligar umidificador"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/Ativa1"
    android:layout_marginStart="15dp"
    android:layout_alignParentTop="true"
    android:layout_alignParentStart="true" />
```

<Button

```
    android:layout_gravity="center_horizontal"
    android:id="@+id/Desativa1"
    android:text="desl. umidificador"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginEnd="12dp"
    android:layout_alignParentTop="true"
    android:layout_alignParentEnd="true" />
```

</RelativeLayout>

<RelativeLayout

```
    android:layout_width="fill_parent"
    android:layout_height="wrap_content">
```

<Button

```
    android:text="Solicitar Temperatura"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/Ativa2"
    android:layout_alignBaseline="@+id/editTemperatura"
    android:layout_alignBottom="@+id/editTemperatura"
    android:layout_alignParentStart="true" />
```

<EditText

```

        android:layout_width="64dp"

        android:layout_height="wrap_content"

        android:inputType="textPersonName"

        android:text="Temp"

        android:ems="10"

        android:id="@+id/editTemperatura"

        android:layout_marginStart="51dp"

        android:layout_alignParentTop="true"

        android:layout_toEndOf="@+id/Ativa2" />
</RelativeLayout>

<RelativeLayout

    android:layout_width="fill_parent"

    android:layout_height="wrap_content">

    <Button

        android:text="Solicitar umidade"

        android:layout_width="wrap_content"

        android:layout_height="wrap_content"

        android:id="@+id/Ativa3"

        android:layout_marginStart="16dp"

        android:layout_alignParentTop="true"

        android:layout_alignParentStart="true" />

    <EditText

        android:layout_width="64dp"

        android:layout_height="wrap_content"

        android:inputType="textPersonName"

        android:text="umid"

        android:ems="10"

        android:id="@+id/editUmidade"

        android:layout_marginEnd="50dp"

        android:layout_centerVertical="true"

```

```

        android:layout_alignParentEnd="true" />
    </RelativeLayout>

    <RelativeLayout

        android:layout_width="fill_parent"

        android:layout_height="wrap_content">

    </RelativeLayout>

    <EditText

        android:layout_width="match_parent"

        android:layout_height="wrap_content"

        android:inputType="textPersonName"

        android:text="Sem detecção de presença"

        android:ems="10"

        android:id="@+id/editPresenca" />
</LinearLayout>

```

// Lista de cores

```

<?xml version="1.0" encoding="utf-8"?>

<resources>

    <color name="Red">#FF0000</color>

    <color name="Green">#93DB70</color>

    <color name="Gray">#DCDCDC</color>

</resources>

```

// Android Manifest

```

<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

    package="com.android.smsdemo" >

    <uses-permission android:name="android.permission.WRITE_SMS" />

```

```

<uses-permission android:name="android.permission.READ_SMS" />
<uses-permission android:name="android.permission.RECEIVE_SMS" />
<uses-permission android:name="android.permission.SEND_SMS" />
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
        android:name=".MainActivity"
        android:label="@string/app_name" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <activity
        android:name=".ReceiveSMSActivity"
        android:label="@string/title_activity_receive_sms" >
    </activity>
    <activity
        android:name=".SendSmsActivity"
        android:label="@string/title_activity_send_sms" >
    </activity>
    <receiver android:name=".SmsBroadcastReceiver"
        android:exported="true" >
        <intent-filter android:priority="999" >
            <action android:name="android.provider.Telephony.SMS_RECEIVED" />
        </intent-filter>
    </receiver>

```

</application>

</manifest>

